NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL (MAVNATT) ARCHITECTURE AND FRAMEWORK

by

Daniel C. McBride

June 2015

Thesis Advisor:	Gurminder Singh
Thesis Co-Advisor:	John Gibson

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2015		**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL (MAVNATT) ARCHITECTURE AND FRAMEWORK			**5. FUNDING NUMBERS**
6. AUTHOR(S) Daniel C. McBride			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Marine Forces Cyber Command			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Tactical networks are becoming more critical in maintaining centers of gravity for military operations as cyberspace becomes contested at all levels of war. As a result, the growth of network centric operations and increased operational tempo in the cyber domain has created a significant training gap for tactical network administrators. This research suggests that a computer-based environment can integrate the operational network and a training network into the same system to allow tactical network administrators to concurrently administer the network and conduct realistic training on an identical virtual network. A review of commercial and open-source tools identifies the baseline for an architecture and framework for this system. The architecture consists of a modular design comprised of mapping, awareness, and virtualization modules. The framework integrates these modules by defining a network topology format, programming language, graphical user interface solution, and virtualization solution. This research concludes by providing an implementation that demonstrates desired capabilities. While we demonstrate that the project goals are attainable, there is a need for further research and development to deploy this capability to fleet units.

14. SUBJECT TERMS network administrator training, network management, network virtualization, tactical network topology, rapid network design, modeling and simulation			**15. NUMBER OF PAGES** 99
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL (MAVNATT) ARCHITECTURE AND FRAMEWORK

Daniel C. McBride
Major, United States Marine Corps
B.S., University of Missouri - Columbia, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2015

Author:	Daniel C. McBride
Approved by:	Gurminder Singh Thesis Advisor
	John Gibson Thesis Co-Advisor
	Peter Denning Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Tactical networks are becoming more critical in maintaining centers of gravity for military operations as cyberspace becomes contested at all levels of war. As a result, the growth of network centric operations and increased operational tempo in the cyber domain has created a significant training gap for tactical network administrators. This research suggests that a computer-based environment can integrate the operational network and a training network into the same system to allow tactical network administrators to concurrently administer the network and conduct realistic training on an identical virtual network. A review of commercial and open-source tools identifies the baseline for an architecture and framework for this system. The architecture consists of a modular design comprised of mapping, awareness, and virtualization modules. The framework integrates these modules by defining a network topology format, programming language, graphical user interface solution, and virtualization solution. This research concludes by providing an implementation that demonstrates desired capabilities. While we demonstrate that the project goals are attainable, there is a need for further research and development to deploy this capability to fleet units.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Program Interface
CUCM	Cisco Unified Communications Manager
DOD	Department of Defense
DODIN	DOD Information Network
EoIP	Everything over Internet Protocol
GIG	Global Information Grid
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ITACS	Information Technology and Communications Services
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KVM	Kernel-Based Virtual Machine
LVC	Live Virtual Constructive
LXC	Linux Containers
MARFORCYBER	Marine Forces Cyber
MAVNATT	Mapping, Awareness, and Virtualization Network Administrator Training Tool
MET	Mission Essential Task
MIB	Management Information Base
MOS	Military Occupational Skill
NETOPS	Network Operations
SDK	Software Development Kit
SNMP	Simple Newark Management Protocol
SOAP	Simple Object Access Protocol
T&R	Training & Readiness
UDP	User Datagram Protocol
UML	User Mode Linux
USMC	United States Marine Corps
VM	Virtual Machine
WMI	Windows Management Instrumentation

XML	Extensible Markup Language
XPCOM	Cross Platform Component Object Model

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Sally, for allowing me to fully invest in this opportunity. I am certain I can never thank her enough for what she has done through her constant support and understanding. I would also like to thank my daughter, Kaylie, who has brought endless inspiration and joy to our family.

I would like to recognize my cohort, faculty, and friends who have provided academic support and encouragement during my time here; you helped the old man knock off some rust. It has been a pleasure to serve with them during this short time.

I would like to express my greatest appreciation and gratitude to my advisors, Dr. Gurminder Singh and John Gibson. Their belief in a vision, guidance toward its accomplishment, and patience during the course of it were immeasurable.

Lastly, I would like to thank Marine Forces Cyber for supporting our endeavor and making this project possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

The Marine Corps' mission as an expeditionary, middleweight, and modern force that is capable of net-centric and cyber-centric operations requires the integration, operation, and management of complex and dynamic networks [1]. Computer networks enable military forces across all functional areas to support command and control, information sharing, and information dominance; they are centers of gravity for military operations and are critical to the success of the military. As the cyber domain becomes more contested and more technically complex, it is critical that network administrators be provided the training and tools to conduct effective network operations (NETOPS), defined as "activities conducted to operate and defend the DOD's Global Information Grid (GIG)" [2]. The joint community executes three foundational cyber missions: DOD information network (DODIN) operations, defensive cyberspace operations (DCO), and offensive cyberspace operations (OCO) [3]. NETOPS and DODIN operations will be the focus of this research. The terms GIG and DODIN refer to the same physical internetwork and are used interchangeably.

Current Marine Corps NETOPS require network administrators to be skilled in many advanced systems, like Microsoft Active Directory, Microsoft Server 2012, Cisco routed networks, VMware virtualization tools, Solar Winds network monitoring tools, network filers, network attached storage, and a multitude of other niche equipment that supports the many different functional areas and operations. Network administrators are rarely offered the opportunity to train on the networks in a live or operational environment. Generally, network administrator activities are limited to responding to actual network outages or critical network modifications deemed necessary to ensure the network is responsive to the training scenario or operation being supported. Specifically, network administrators cannot train to tactics, techniques, and procedures on the live network as it could jeopardize the availability of the network to users. Furthermore, network administrators do not have simple and lightweight tools to manage the current network, as well as project how network changes may affect the live network.

Network administrators can no longer be viewed simply as a support-only function to training events and operation. The criticality of communications and network operations demand the same training and operational capabilities as combat arms specialties in order to assess the impact of events on network operations and provide a platform by which the network administrators may be evaluated to the same degree as those of the other combat arms in live training or operational environments.

Our research will investigate and prototype an environment that is capable of providing network administrators the ability to automatically map the network topology, maintain network awareness of components comprising that topology, and virtualize all or a portion of that topology for training and evaluation. The guidance for this environment will be that it is simple, lightweight, responsive, and resilient so it can operate across the spectrum of operations and in highly dynamic environments, specifically targeting the portion of the operational network under the oversight and operation of those being trained or evaluated. This research will concentrate on small-scale tactical networks commonly found at the regiment/brigade level and below.

B. MOTIVATION

The motivation for this thesis derives from the author’s 14 years of experience provisioning, installing, operating, and maintaining tactical networks in the Marine Corps. Military garrison networks are very similar to commercial enterprise networks and utilize standard network operation and training methodologies. However, these common standards do not translate well to tactical networks. A tactical network is an integral part of its unit and must be capable of being moved, operated, and changed based on the needs of the unit to accomplish the unit’s mission. Procedurally, garrison and tactical network operations may be similar, but the underlying equipment, environment, and operational context to support a tactical network are dynamic and complex. Similarly, there is a gap in network training between garrison and tactical network administrators.

Garrison network training is conducted in a stable environment where equipment, space, and time are not major limitations. Training can be facilitated on simulators, parallel architectures, or on fully virtualized architectures like the DOD cyber and information assurance ranges [4], [5].

Tactical networks are the complete opposite. Training equipment, space, and time are rarely available due to the tactical unit's primary mission and limited embarkation capabilities. Tactical network administrators do not have the luxury of establishing parallel or fully virtualized architectures, let alone the capacity to haul the additional equipment necessary to do so. Operational tempo also reduces the time available for training, especially when a unit may setup and teardown the entire tactical infrastructure multiple times per day. This tactical network administrator training is a critical gap.

Preliminary research identifies three functional areas required to support network administration in a tactical environment: mapping, awareness, and virtualization. A single network administrator system that addresses all three functional areas could mitigate this training gap.

C. SCOPE

The primary scope of this thesis is to establish the framework and architecture of a mapping, awareness, and virtualization network administrator training tool (MAVNATT), thereby facilitating cohesive, follow-on research. It is important to identify that MAVNATT is intended to be a lightweight system, both figuratively and literally, to support network administrators in tactical environments. It is not intended to replace the existing DOD/Service cyber ranges, nor is it intended to replace the robust commercial applications currently used to support network operations at higher or more stable military network operation centers.

D. RESEARCH OBJECTIVE

The primary research objective this thesis endeavors to achieve is to demonstrate that a MAVNATT system is attainable through development of an architecture and framework leading to the specification of an operational prototype that is capable of providing network administrators a computer-assisted integrated ability to map the network topology, maintain network awareness of that topology, and virtualize all or a portion of that topology for training and evaluation.

E. EXPLORATORY RESEARCH QUESTIONS

The following research questions lead to the achievement of the research objective:

1. What are suitable formats to represent a network topology for importing, exporting, and sharing?

2. What are suitable libraries and application program interfaces (APIs) that can be utilized to create a graphical user interface (GUI) based system that is capable of visually representing physical and virtual networks?

3. What are the methods of dynamically modeling a physical network to form a virtual network?

4. What are the methods of modeling network devices in a virtual environment?

5. Can this system be used to demonstrate scenario-based training?

6. Can this system work with current DOD tools?

F. ASSUMPTIONS

A generic baseline must be assumed in order to begin this research. The following assumptions have been made in order to utilize research efforts efficiently and effectively and to produce a capable prototype.

Network Topology. MAVNATT must have knowledge of the network. This is assumed because the research goal is to assist network administrators with mapping, providing awareness, and virtualizing an existing network. Therefore, a certain understanding, knowledge, and permission base must exist. This assumption prevents the researching of extremely niche areas and methodologies that could detract from the overall research goal.

Open Standards. MAVNATT will focus on the utilization of open standards and systems. This is assumed in order to ensure application independence, platform independence, and long-term access to resources. This assumption prevents researchers from being locked out of understanding solutions, and it will assist in rapidly deploying an operational prototype.

G. APPROACH

We will explore technologies to determine suitable components to establish an architecture and framework for MAVNATT. After identifying the architecture modules, framework components will be identified and prototypes designed to support the integration of the modules. Finally, an integrated prototype will be designed that demonstrates properties desired in follow-on research and module development.

H. BENEFITS OF RESEARCH

The overarching MAVNATT project and research has the potential to demonstrate that a lightweight system can be utilized in tactical environments to provide training and evaluation capabilities to tactical network administrators without jeopardizing the unit mission and without jeopardizing the network on which users are operating. This thesis supports the primary project by identifying the training gap that exists for network administrators operating in tactical environments, and by identifying an architecture and framework for the MAVNATT system.

I. ORGANIZATION

This thesis is organized as follows:

Chapter II: Background. This chapter identifies the training gap that exists for tactical network administrators. It then provides a summary of industry solutions used for network administrator operations and training that addresses each of the defined functional areas. Lastly, it reduces the classification of applications into two categories, network monitoring suites and network simulators, and demonstrates that these categories of applications do not cover the spectrum of task MAVNATT is intended to address.

Chapter III: Design and Methodology. This chapter identifies the desired architecture and framework for the MAVNATT system. It demonstrates that the architecture modules align with the defined functional areas. It then demonstrates framework components that are required to integrate the modules.

Chapter IV: Implementation. This chapter will demonstrate examine the implementation concepts of MAVNATT by demonstrating the architecture and framework with an integrated prototype.

Chapter V: Conclusions and Future Research. This chapter summarizes the thesis effort, provides conclusions to the research questions, and provides recommendations for future research, specifically, advancing the research fidelity within each framework component and architecture module.

II. BACKGROUND

A. OVERVIEW

The intent of MAVNATT is to provide a lightweight system that can be used to support network administrator training and operations. Preliminary research identifies three functional areas that are critical to tactical network administrators: mapping, awareness, and virtualization. This conceptual model of a MAVNATT system is depicted in Figure 1. Mapping, or network auto-discovery, allows a network administrator to know the physical layout and composition of the network. Awareness, or network monitoring, allows a network administrator to understand the status and capability of the network. Virtualization provides an environment to train and evaluate administrator performance that does not harm the operational network. We feel a solution that supports these functional areas will greatly benefit network administrators in tactical environments.

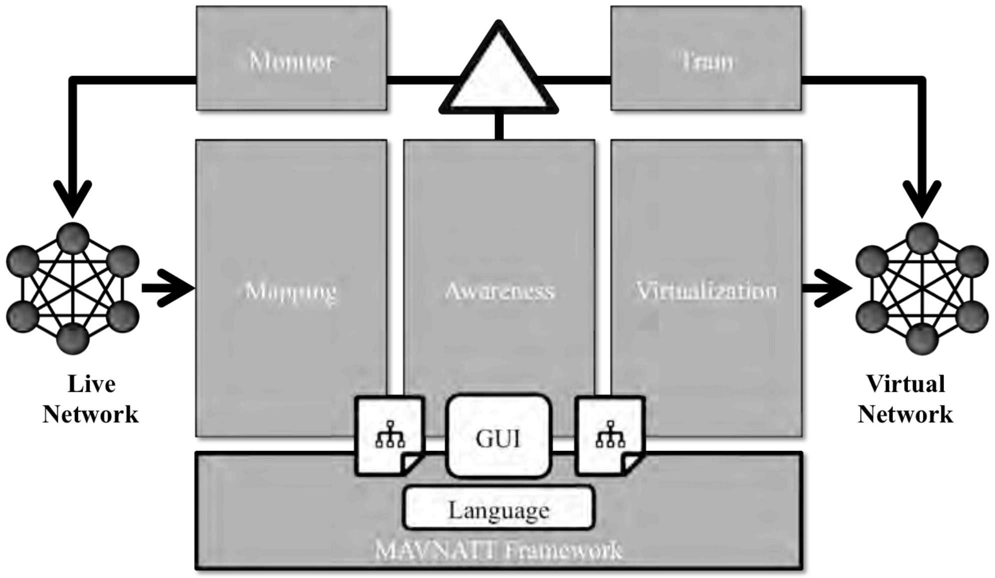


Figure 1. MAVNATT Conceptual Model

This chapter presents an overview of the current military network administrator training at the tactical level of a military unit. It then provides a survey of the current tools used by industry in each functional area of MAVNATT. It also surveys network simulators, which is a hybrid of these functional areas.

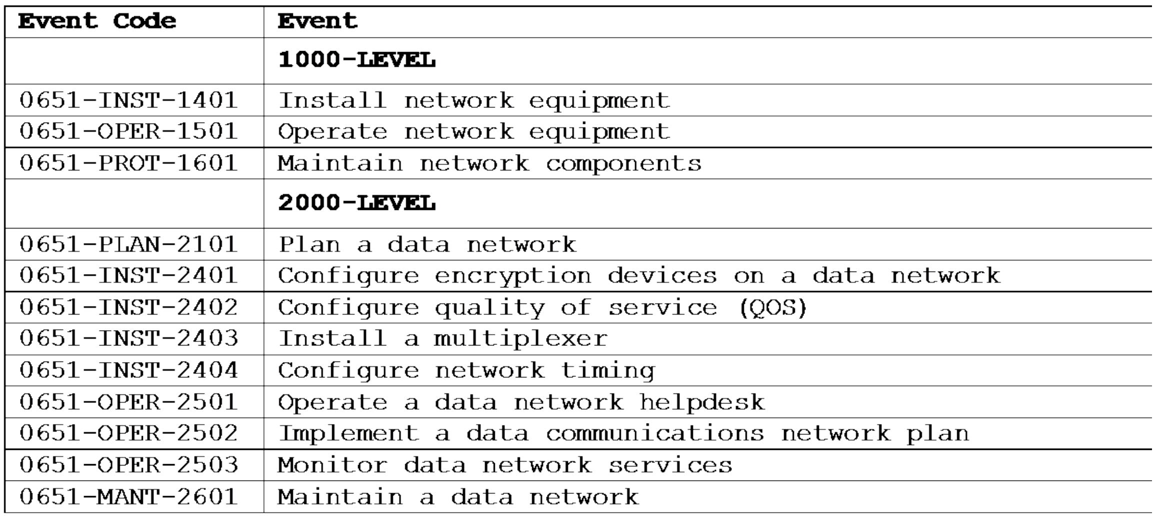
B. NETWORK ADMINISTRATOR TRAINING

In order to better understand the value of MAVNATT as a training tool, we must first understand the training of network administrators and the tasks they are expected to conduct in tactical environments.

The *United States Marine Corps (USMC) Military Occupational Specialties Manual (MOS) Manual* [6] and *Communication Training & Readiness (T&R) Manual* [7] provide guidance with respect to the requirements for a trained USMC MOS 0651, Cyber Network Operator. Though these requirements are not consistent across the DOD, they provide a reference point to describe military network administrator training.

1. Required Training

MOS training, or required training, is where core skill training is provided to the individual Marine, normally at an MOS-producing formal school course. The *MOS Manual* states that designated Marines for cyber network operator, or 0651s, must complete two courses at the Marine Corps Communications and Electronics School (MCCES), located at Marine Corps Base Twenty-nine Palms, California: M09CGW1 Information Technology (IT) Essentials and M09BNW1 Cyber Network Specialist. The outcome of these courses provides a baseline-trained network specialist that is capable of the 1000-level individual events outlined in Figure 2.



Event Code	Event
	1000-LEVEL
0651-INST-1401	Install network equipment
0651-OPER-1501	Operate network equipment
0651-PROT-1601	Maintain network components
	2000-LEVEL
0651-PLAN-2101	Plan a data network
0651-INST-2401	Configure encryption devices on a data network
0651-INST-2402	Configure quality of service (QOS)
0651-INST-2403	Install a multiplexer
0651-INST-2404	Configure network timing
0651-OPER-2501	Operate a data network helpdesk
0651-OPER-2502	Implement a data communications network plan
0651-OPER-2503	Monitor data network services
0651-MANT-2601	Maintain a data network

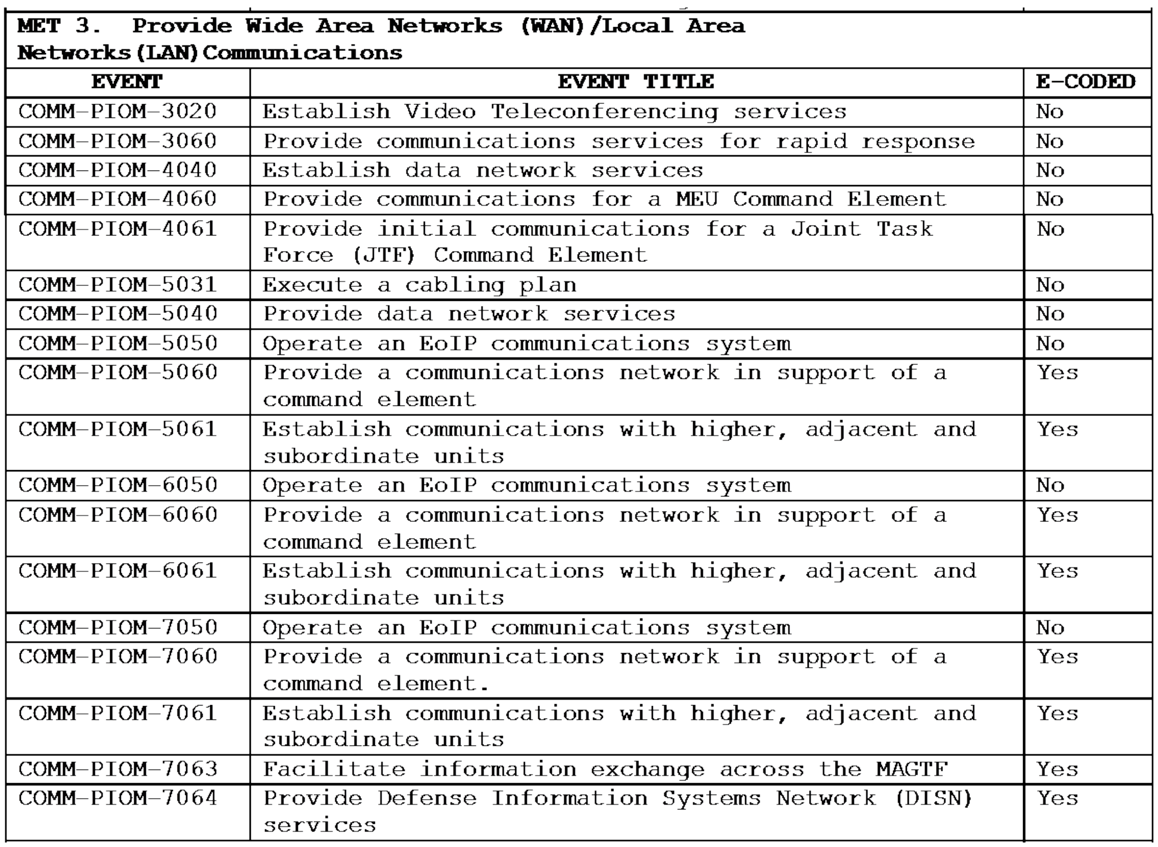
Figure 2. Individual 0651 Training Events, from [7]

2. Skill Progression/Enhancement Training

The USMC provides follow-on training to increase overall skills or knowledge. Marine 0651 follow-on training is provided solely through MCCES at resident courses in 29 Palms and the MCCES-run communication training centers (CTCs) located at each Marine Expeditionary Force (MEF). These schools provide classes covering the 2000-level individual events outlined in Figure 2. This training is provided at the discretion of the unit responsible for the individual 0651 and is limited by the availability of classes or seats.

3. Training & Readiness Individual and Collective Events

Marine 0651s must also train to all collective events for their unit, which are found in the T&R manuals for the associated unit (i.e., infantry, artillery, logistics). These collective events are vague and often fail to specifically identify the individual tasks associated with the collective event, which can prevent 0651s from being properly trained or evaluated during unit level collective training. Concurrent to the unit collective training, all 0651 are expected to be able to execute 1000- and 2000-level individual events, regardless of their rank and whether or not they have received the actual training. This is normally due to the quantity and rank of 0651s assigned to units and not a function of the unit attempting to disregard the *MOS Manual* and T&R manuals. Figure 3 provides an example of collective events for a communication battalion.



MET 3. Provide Wide Area Networks (WAN)/Local Area Networks(LAN)Communications

EVENT	EVENT TITLE	E-CODED
COMM-PIOM-3020	Establish Video Teleconferencing services	No
COMM-PIOM-3060	Provide communications services for rapid response	No
COMM-PIOM-4040	Establish data network services	No
COMM-PIOM-4060	Provide communications for a MEU Command Element	No
COMM-PIOM-4061	Provide initial communications for a Joint Task Force (JTF) Command Element	No
COMM-PIOM-5031	Execute a cabling plan	No
COMM-PIOM-5040	Provide data network services	No
COMM-PIOM-5050	Operate an EoIP communications system	No
COMM-PIOM-5060	Provide a communications network in support of a command element	Yes
COMM-PIOM-5061	Establish communications with higher, adjacent and subordinate units	Yes
COMM-PIOM-6050	Operate an EoIP communications system	No
COMM-PIOM-6060	Provide a communications network in support of a command element	Yes
COMM-PIOM-6061	Establish communications with higher, adjacent and subordinate units	Yes
COMM-PIOM-7050	Operate an EoIP communications system	No
COMM-PIOM-7060	Provide a communications network in support of a command element.	Yes
COMM-PIOM-7061	Establish communications with higher, adjacent and subordinate units	Yes
COMM-PIOM-7063	Facilitate information exchange across the MAGTF	Yes
COMM-PIOM-7064	Provide Defense Information Systems Network (DISN) services	Yes

Figure 3. Sample of Collective Training Events, from [7]

4. Network Administrator Training in Tactical Environments

Units conduct block training to meet the unit's Mission Essential Tasks (METs), associated with the unit T&R manual. This training normally progresses from individual to collective events over the course of three to six months, depending on unit operational tempo and deployment cycles. During this training, a network administrator may be in a tactical environment for multiple five-day training evolutions covering 1000- to 5000-level events, which are individual to platoon level events. Figures 2 and 3 provide examples of the tasks expected to be conducted, such as "0651-INST-1403 - Install network equipment," "0651-OPER-2503 - Monitor Data Services," and "COMM-PIOM-5040 - Provide data network services." These tasks are very vague and do not provide the detail and requirements that support network administrator training.

During the same six-month training period, the same network administrator may participate in an additional 30-day training evolution covering 5000- to 8000-level events, which includes platoon through regimental level events. These tasks include "COMM-PIOM-5050 - Operate and EoIP communications system," "COMM-PIOM-6050 - Operate and EoIP communications system," and "COMM-PIOM-7050 - Operate and EoIP communications system," where EoIP means everything over Internet protocol. These tasks are broad, redundant, and cannot easily be decomposed into subordinate tasks for network administrator training.

During all these individual and collective events, the unit and supported personnel require the network to be operational for the duration of the training evolution. This often means that 0651s get minimal training opportunities, relegated only to the required installation, maintenance, and teardown of the network. Any training opportunities beyond these required tasks could interfere with the stability of the network and are therefore not authorized.

5. Network Administrator Training Summary

We can see from these 1000- and 2000-level events that a network administrator is expected to be broadly trained and extremely capable in support of his unit. The specific task, standard, and conditions outlined in the T&R manuals do not provide designated tools for the support of network operations and training, nor methods of evaluation of the skills the 0651. Therefore, there is a significant gap in the ability to train across the full complement of tasks and events directed in accordance with the *MOS Manual* and T&R manuals. This training gap is what MAVNATT is attempting to correct, by allowing 0651s, or network administrators, to be able to conduct operations and training on the same network topology without risk to its operation.

C. CURRENT TOOLS SURVEY

As discussed in Section B, there are no tools designated by the T&R Manual for the operations and training of network administrators. Consequently, many units purchase commercial-off-the-shelf tools to support operations, but they do not correspondingly purchase tools that support network administrator training. More interestingly, there does not appear to be a tool that completely integrates network operations and training.

The following sections survey tools currently in use by industry and academic institutions in order to identify capabilities that are critical to tactical level network operations and training; such tools are considered to provide components for the MAVNATT prototype. Our criteria to be utilized in the survey include whether (1) the tools were utilized in the academic courses of instruction, (2) the tools were utilized in current DOD network domains, or (3) the tools were identified as predominant leads in their domain through multiple Internet searches. We have broken this survey down into the three identified functional areas: mapping, awareness, and virtualization. We also survey network simulators.

1. Mapping

Network mapping is the process of discovering devices and their associated connectivity on a network. As networks become more complicated and dynamic, network administrators require the support of automated mapping tools to verify their current topology and to identify possible rogue devices or services that pose a security risk to the network. The availability of automated network mapping tools improves the effectiveness and efficiency in the tactical domain by allowing network administrators to identify the network topology, compare it to a planned topology, or identify changes to a topology so the changes can be validated.

In a tactical environment, the ability to quickly and effectively verify a network plan also minimizes the personnel requirement and workload involved in troubleshooting activities during the establishment and maintenance phases of a network. Network maps can increase a network administrator’s situational awareness by providing a complete and accurate view of the network in order to maintain availability and security of the network. Due to the constant state of network flux resulting from the need to establish and teardown the network multiple times per day, equipment faults and manual topology changes occur at a very high rate. Network maps allow network administrators to respond quickly to these faults and changes.

This section discusses the Internet Protocol Suite, simple network management protocol (SNMP), and two applications that utilize basic protocol characteristics for mapping a network topology, Angry IP Scanner and NMap.

a. Internet Protocol Suite

The Internet Protocol Suite (also referred to as the TCP/IP Protocol Suite) consists of the core network protocol standards that provide the underlying architecture for all networks [8]. These same protocols provide the basis for all network management, mapping, monitoring, and awareness tools. The Internet Protocol Suite is graphically described in Figure 4.

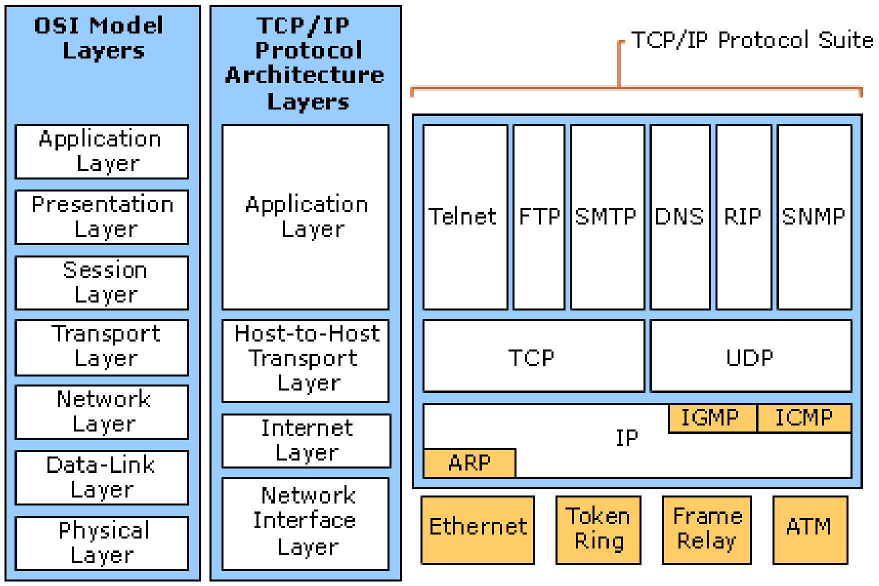


Figure 4. Internet Protocol Suite, from [9]

The protocols defined in the Internet Protocol Suite have specific characteristics that they are expected to demonstrate during the normal or failed operations in support of the transportation of data, intercommunication of devices, and the management of networked devices. These automated protocol communications and responses allow networks to operate at extreme speeds with minimal user interaction. Network administrators can use these protocol characteristics to infer certain attributes about the network, such as latency and hops between devices. The majority of the protocols operate automatically among network-connected devices; therefore, utilities were developed to interact with the protocols for the purposes of retrieving user accessible information.

Utilities such as ping and traceroute exploit these protocol characteristics in order to measure specific aspects of the network [10]. Ping can determine possible existence of a device on the network and the latency to that device based on round trip times. Traceroute can determine the existence and availability of a route to a device and the transit delays through devices. These measurements are only available if the devices are properly configured to support the underlying protocols. For example, ping and traceroute depend on the operational availability of the Internet Control Message Protocol (ICMP) on the devices involved in the execution of these applications.

Other utilities, such as Paris traceroute and Rocketfuel, were designed in order to mitigate specific protocol anomalies due to network characteristics, such as flow-based load balancing and Internet protocol (IP) ID counters [11]. These protocol utilities provide the foundation for the other mapping techniques and technologies that we survey; this is why they deserve their own discussion. No single protocol can individually map the network, which is why most techniques utilize multiple protocols to infer a network topology. Most operating systems provide access to these protocols, but they do not provide the ability to import/export topologies based on the determination of the protocol. Manual interpretation and transcription can be used, but this is very time consuming. For these reasons, many utilities and applications are created to leverage these protocols for the purposes of network mapping.

b. Simple Network Management Protocol (SNMP)

SNMP is part of the Internet Protocol suite that exists at the application level. It is specifically designed for the management and monitoring of IP networks. SNMP consists of a set of standards for network management, a database schema, and a set of data objects [12]. SNMP provides access to management data on the managed systems that can then be queried or set by managing applications. Industry uses SNMP to monitor network availability, performance, and error rates. SNMP provides an excellent capability to derive network topologies because managing applications can get network interface information, including connected devices, directly from SNMP-managed devices. SNMP is available on all OSs including Windows, Mac OS X, and Linux. It is also available on Cisco routers, Cisco Unified Communications Manager (CUCM), and IP phones. A secure version of SNMP, or SNMPv3, is not as widely distributed or implemented in network architectures [13]. Figure 5 demonstrates the SNMP model with interactions between the SNMP manager and agents.

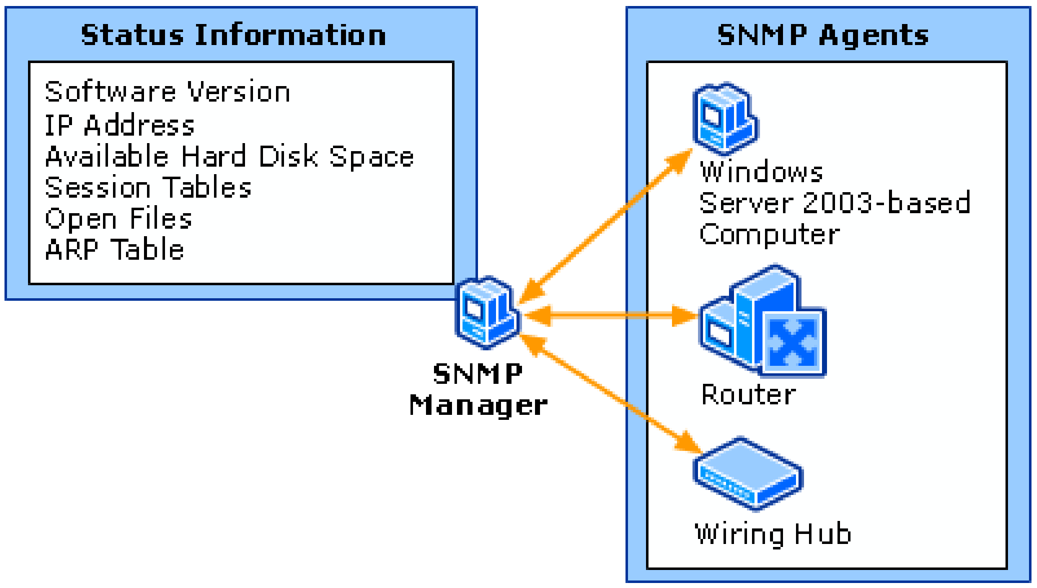


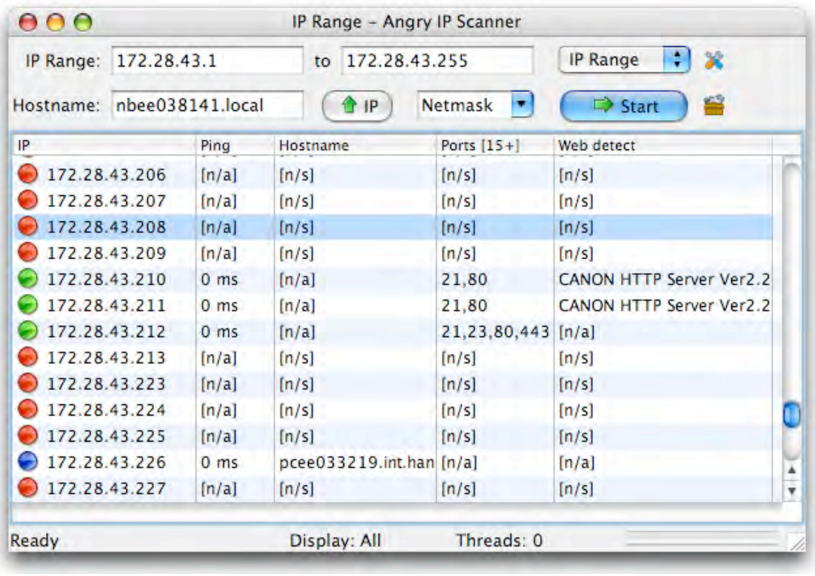
Figure 5. SNMP Model, from [14]

SNMP uses a defined architecture consisting of SNMP Managers, SNMP Agents, and Management Information Bases (MIBs) to manage devices within a community. SNMP managers have the ability to collect information from SNMP Agents that are defined in a MIB. Each element of information in a MIB has an Object Identifier (OID) that is in Abstract Syntax Notation One (ASN.1). The SNMP Manager can query SNMP agents through predefined protocol language commands that consist of GET, GET-NEXT, SET, GET-BULK, and NOTIFY. SNMP Agents can push information to the SNMP Manger through a trap, utilizing the NOTIFY command [13]. This simple architecture and language allows network administrators to maintain a very high level of visibility and control over the network. A very accurate picture of the network topology can be learned through standard SNMP MIBs.

SNMP requires manual configuration of all devices on the network in order to provide SNMP Managers full visibility of the SNMP Agents and MIB data. Though this manual process could be integrated into unit procedures, it does not scale well to environments where joint or combined operations are conducted or where network devices are added and removed from tactical networks based on operational requirements. These additional systems may not be guaranteed to be running SNMP, and some devices may not have administrative permissions to allow network administrators to enable SNMP. There are also security concerns because not all devices have SNMPv3 capability; using SNMPv1/v2 creates severe security vulnerabilities to the network. This issue suggests that SNMP should not be utilized as the primary mapping tool; however, SNMP should be considered for follow-on research to integrate it into MAVNATT.

c. Angry IP Scanner

Angry IP Scanner is an open-source and cross-platform network scanner designed to be fast and simple to use. It is a Java-based application that can scan individual or ranges of IP addresses, as well as ports and protocols [15], [16]. This application has a breadth of peer applications that will not be discussed, as they are all similar in capability and design. Angry IP Scanner is very simple in its implementation and utilizes the Internet IP Suite protocols and utilities to derive its information. Figure 6 demonstrates the Angry IP Scanner interface.



IP Range: 172.28.43.1 to 172.28.43.255

Hostname: nbee038141.local Netmask

IP	Ping	Hostname	Ports [15+]	Web detect
🔴 172.28.43.206	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.207	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.208	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.209	[n/a]	[n/s]	[n/s]	[n/s]
🟢 172.28.43.210	0 ms	[n/a]	21,80	CANON HTTP Server Ver2.2
🟢 172.28.43.211	0 ms	[n/a]	21,80	CANON HTTP Server Ver2.2
🟢 172.28.43.212	0 ms	[n/a]	21,23,80,443	[n/a]
🔴 172.28.43.213	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.223	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.224	[n/a]	[n/s]	[n/s]	[n/s]
🔴 172.28.43.225	[n/a]	[n/s]	[n/s]	[n/s]
🔵 172.28.43.226	0 ms	pcee033219.int.han	[n/a]	[n/a]
🔴 172.28.43.227	[n/a]	[n/s]	[n/s]	[n/s]

Ready Display: All Threads: 0

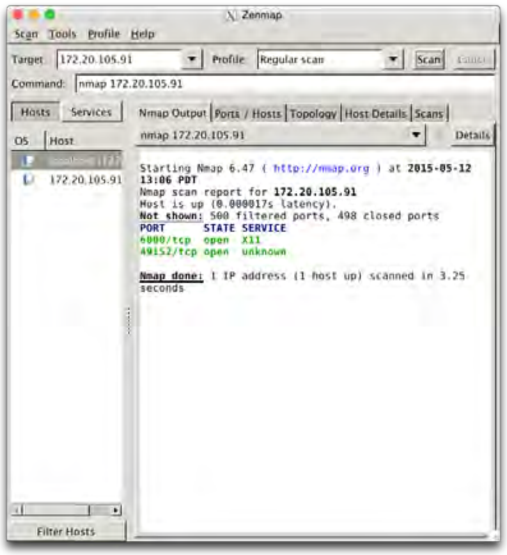
Figure 6. Angry IP Scanner Interface on Mac OS X, from [15]

Conceptually, the methods and techniques used by the Angry IP Scanner and similar applications can be integrated into MAVNATT to support both mapping and awareness functional areas. Angry IP Scanner uses a brute force processes to determine if a network host responds to standard protocol requests. In most cases, unless a client is setup to explicitly deny these requests, there will be some level of response to indicate if the host is active and reachable. Angry IP Scanner and similar applications do not organically infer the network topology because they do not directly assess what network devices are connected to each other. These applications provide information such as availability and what devices are on the same broadcast domain, which can be used for mapping inferences.

d. Nmap/Zenmap.

Nmap is an open source network mapping tool that can be used by network administrators to discover or verify network topology, identify network services that are available, identify host operating systems, and identify other network protocol characteristics. Nmap provides a broader scan of network protocols and services than Angry IP Scanner. Nmap is a command line tool, although a graphical user interface called Zenmap is available to simplify Nmap usability. Figure 7 demonstrates the

Zenmap interface. These tools were written in C++ and designed to work on large-scale networks. They can save an extensible markup language (XML) file that contains all the results from the scan for review. The Zenmap interface also provides a topology map that can be saved as an image [17], [18]. Nmap/Zenmap and its techniques should be investigated further to see if they can provide relevant data for the mapping portion of MAVNATT.



```
Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-12
13:06 PDT
Nmap scan report for 172.20.105.91
Host is up (0.000017s latency).
Not shown: 500 filtered ports, 498 closed ports
PORT      STATE SERVICE
6000/tcp  open  X11
49152/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 3.25
seconds
```

Figure 7. Zenmap Interface on Mac OS X

e. Mapping Summary

This section briefly described the standard protocols of the Internet Protocol suite and how the default protocol characteristics form the basis of most mapping and monitoring solutions, it further explained SNMP and its applicability to network mapping as well as security concerns, and lastly it discussed two tools that demonstrate capabilities required by MAVNATT in its ability to map a network topology and describe it into a standardized format. The next section discusses network awareness and monitoring tools. Some of these tools have integrated mapping capabilities that are specific to that application and further advance mapping techniques that could be useful in MAVNATT.

2. Awareness

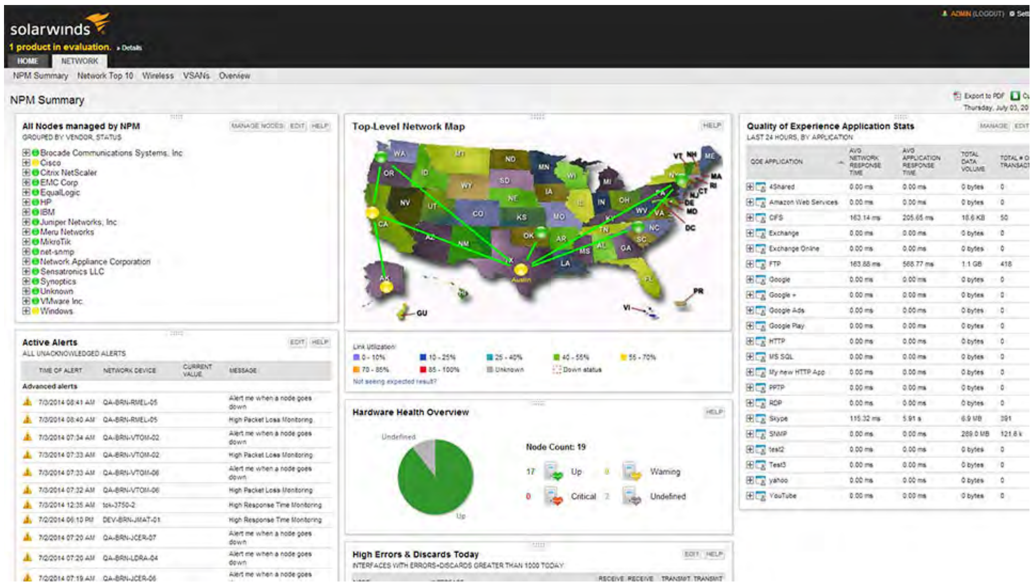
Network awareness is the process of providing a timely and relevant assessment of the network to the network administrator. In the context of this thesis, network awareness falls under the larger umbrellas of network management and network monitoring. Network management is classified into five areas by the ISO Telecommunications Management Network framework: Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management [19]. Network monitoring is a function that supports network management by collecting information from network devices to specifically address performance, fault, and accounting management areas [13]. The intended goal of network awareness with respect to MAVNATT aligns with fault management and fault monitoring. Specifically, network awareness intends to provide visualization of the network topology and fault detection capabilities in order to enhance situational awareness to the network administrator.

Network awareness is critical for network administrators in both garrison and tactical environments. These network administrators need quick and reliable status updates of the devices of their network throughout the network tasks of establishment, operations, and teardown. In tactical operations, the cycle of these tasks is much more frequent than civilian or garrison counterparts. Similar to the requirement for network maps, network awareness allows network administrators to respond quickly to network faults and changes.

This section discusses the common industry network management suites that provide network monitoring and network awareness capabilities. In general, network management suites have their own version of a mapping tool that will be briefly mentioned for each suite, but these tools do not differ substantially from the tools identified in the previous section and will not be discussed in detail.

a. SolarWinds Network Performance Manager

SolarWinds Network Performance Manager (NPM) is the only network-monitoring tool authorized on USMC networks, which is why it is considered in this survey. SolarWinds NPM is a commercial grade network management suite that provides a full-featured monitoring system. It is primarily built upon SNMP and Windows Management Instrumentation (WMI) protocols for administration and features a web-based visualization interface that provides real-time monitoring and statistics for network management. Figure 8 demonstrates the SolarWinds NPM interface. SolarWinds has many advanced features that provide network administrators with a clear understanding of all aspects of their network. It also allows for customizable interfaces and reports. Despite SolarWinds NPM complexity, it is still an intuitive program to use [20].



NPM Summary

All Nodes managed by NPM
GROUPED BY VENDOR, STATUS

- Brocade Communications Systems, Inc
- Cisco
- Citrix NetScaler
- EMC Corp
- EqualLogic
- HP
- IBM
- Juniper Networks, Inc
- Meru Networks
- MikroTik
- net-snmp
- Network Appliance Corporation
- Sensatronics LLC
- Synoptics
- Unknown
- VMware Inc
- Windows

Active Alerts
ALL UNACKNOWLEDGED ALERTS

TIME OF ALERT	NETWORK DEVICE	CURRENT VALUE	MESSAGE
Advanced alerts			
7/3/2014 08:41 AM	QA-BRN-RMEL-05		Alert me when a node goes down
7/3/2014 08:40 AM	QA-BRN-RMEL-05		High Packet Loss Monitoring
7/3/2014 07:34 AM	QA-BRN-VTOM-02		Alert me when a node goes down
7/3/2014 07:33 AM	QA-BRN-VTOM-02		High Packet Loss Monitoring
7/3/2014 07:33 AM	QA-BRN-VTOM-06		Alert me when a node goes down
7/3/2014 07:32 AM	QA-BRN-VTOM-06		High Packet Loss Monitoring
7/3/2014 12:35 AM	tek-3750-2		High Response Time Monitoring
7/2/2014 06:10 PM	DEV-BRN-JMAT-01		High Response Time Monitoring
7/2/2014 07:20 AM	QA-BRN-JCER-07		Alert me when a node goes down
7/2/2014 07:20 AM	QA-BRN-LDRA-04		Alert me when a node goes down
7/2/2014 07:19 AM	QA-BRN-JCER-06		Alert me when a node goes

Link Utilization
🟪 0 - 10%	🟦 10 - 25%	🟦 25 - 40%	🟩 40 - 55%	🟨 55 - 70%
🟧 70 - 85%	🟥 85 - 100%	⬜ Unknown	⬚ Down status	

Hardware Health Overview

Node Count: 19

17	Up	0	Warning
0	Critical	2	Undefined

High Errors & Discards Today
INTERFACES WITH ERRORS+DISCARDS GREATER THAN 1000 TODAY

Quality of Experience Application Stats
LAST 24 HOURS, BY APPLICATION

QOE APPLICATION	AVG NETWORK RESPONSE TIME	AVG APPLICATION RESPONSE TIME	TOTAL DATA VOLUME	TOTAL # OF TRANSACT
4Shared	0.00 ms	0.00 ms	0 bytes	0
Amazon Web Services	0.00 ms	0.00 ms	0 bytes	0
CIFS	163.14 ms	205.65 ms	18.6 KB	50
Exchange	0.00 ms	0.00 ms	0 bytes	0
Exchange Online	0.00 ms	0.00 ms	0 bytes	0
FTP	163.88 ms	568.77 ms	1.1 GB	418
Google	0.00 ms	0.00 ms	0 bytes	0
Google +	0.00 ms	0.00 ms	0 bytes	0
Google Ads	0.00 ms	0.00 ms	0 bytes	0
Google Play	0.00 ms	0.00 ms	0 bytes	0
HTTP	0.00 ms	0.00 ms	0 bytes	0
MS SQL	0.00 ms	0.00 ms	0 bytes	0
My new HTTP App	0.00 ms	0.00 ms	0 bytes	0
PPTP	0.00 ms	0.00 ms	0 bytes	0
RDP	0.00 ms	0.00 ms	0 bytes	0
Skype	115.32 ms	5.91 s	6.9 MB	391
SNMP	0.00 ms	0.00 ms	289.0 MB	121.6 k
test2	0.00 ms	0.00 ms	0 bytes	0
Test3	0.00 ms	0.00 ms	0 bytes	0
yahoo	0.00 ms	0.00 ms	0 bytes	0
YouTube	0.00 ms	0.00 ms	0 bytes	0

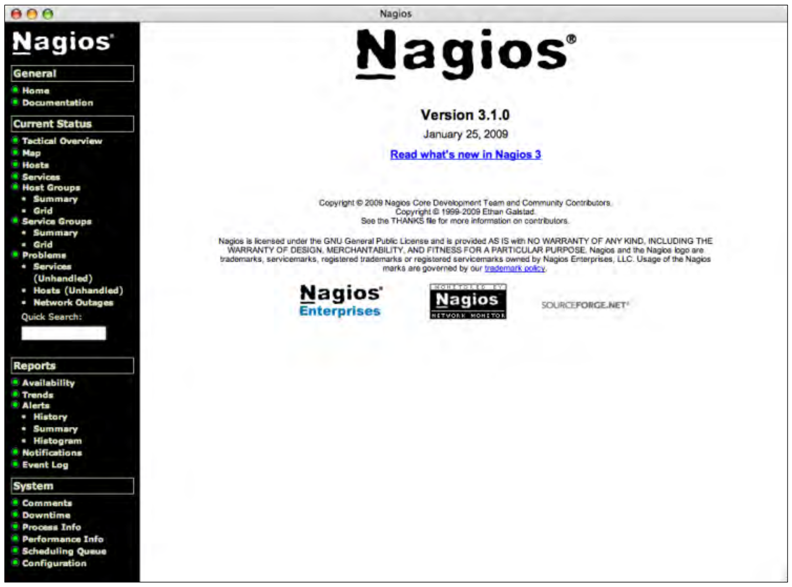
Figure 8. SolarWinds NPM 11.5 Interface, from [21]

SolarWinds NPM meets all expectations to provide situational awareness for network administrators. In addition to its network performance and audit monitoring, it also provides auto-discovery and auto-configuration capabilities as long as the network devices have been configured for SNMP or WMI. SolarWinds NPM utilizes Network Sonar, its integrated mapping tool, to accomplish this task. Network Sonar utilizes ICMP ping as well as credential-based SNMPv1/v2/v3, WMI, and VMware protocols for its auto-discovery and mapping capability [22].

SolarWinds NPM is a powerful solution for network administration, management, and monitoring. It is heavily integrated with Microsoft and VMware products, both of which have strong positions in the DOD. SolarWinds NPM is far from a lightweight solution, requiring a Windows 2003 or greater server with Microsoft SQL server installed. Also, the SolarWinds pricing structure is based on network "elements" being monitored, starting with 100 elements at $2795.

b. Nagios

The Naval Postgraduate School Information Technology and Communications Services (ITACS) department utilizes Nagios for their network monitoring solution. Nagios is an open source network monitoring solution that was written in the C programming language and runs on Linux and Unix based systems. All provisioning is based around the Nagios Core application, a powerful backend solution for network management. Nagios XI is a commercial offering that is built around the Nagios Core and provides a more refined interface and better usability. Nagios Core and Nagios XI are both extendable with plugins that provide many network management tools including frontend solutions and network discovery. This modular approach makes Nagios very flexible and scalable [23]-[25]. Figure 9 demonstrates the Nagios web-based interface.



Version 3.1.0

January 25, 2009

Read what's new in Nagios 3

Copyright © 2009 Nagios Core Development Team and Community Contributors.
Copyright © 1999-2009 Ethan Galstad.
See the THANKS file for more information on contributors.

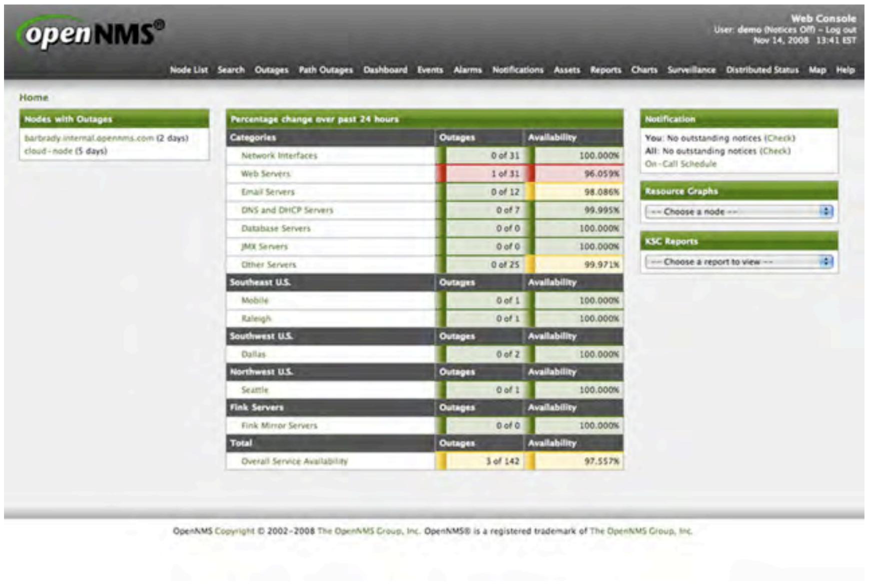
Nagios is licensed under the GNU General Public License and is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Nagios and the Nagios logo are trademarks, servicemarks, registered trademarks or registered servicemarks owned by Nagios Enterprises, LLC. Usage of the Nagios marks are governed by our trademark policy.

Figure 9. Nagios Interface, from [26]

Nagios Core and Nagios XI are as capable as other commercial systems for network monitoring, and it is more cost effective. Some limitations of Nagios include its reliance on a Linux/Unix-based host system since Linux/Unix are difficult to inject into the DOD networks due to the limited training of network administrators on Unix-based systems. The system is also antiquated and in need of a systematic overhaul to continue to compete with commercial systems.

c. OpenNMS

OpenNMS is another open source network-monitoring solution that is utilized in industrial and academic environments. It is written in the Java programming language and is capable of running on all major operating systems. OpenNMS is capable of all network monitoring areas and can also perform integrated network discovery with SNMP, LDAP, ICMP, SSH, and other protocols [24], [27], [28]. Figure 10 demonstrates the OpenNMS interface.



Nodes with Outages

barbrady.internal.opennms.com (2 days)
cloud-node (5 days)

Percentage change over past 24 hours

Categories	Outages	Availability
Network Interfaces	0 of 31	100.000%
Web Servers	1 of 31	96.059%
Email Servers	0 of 12	98.086%
DNS and DHCP Servers	0 of 7	99.995%
Database Servers	0 of 0	100.000%
JMX Servers	0 of 0	100.000%
Other Servers	0 of 25	99.971%
Southeast U.S.	**Outages**	**Availability**
Mobile	0 of 1	100.000%
Raleigh	0 of 1	100.000%
Southwest U.S.	**Outages**	**Availability**
Dallas	0 of 2	100.000%
Northwest U.S.	**Outages**	**Availability**
Seattle	0 of 1	100.000%
Fink Servers	**Outages**	**Availability**
Fink Mirror Servers	0 of 0	100.000%
Total	**Outages**	**Availability**
Overall Service Availability	3 of 142	97.557%

Notification

You: No outstanding notices (Check)
All: No outstanding notices (Check)
On-Call Schedule

OpenNMS Copyright © 2002-2008 The OpenNMS Group, Inc. OpenNMS® is a registered trademark of The OpenNMS Group, Inc.

Figure 10. OpenNMS Interface, from [27]

OpenNMS is as capable as the other reviewed network monitoring tools. It provides more usability to the network administrator than Nagios [24]. It is also more capable of being utilized on DOD networks because it can be run on a Windows-based platform. Its largest drawback is that it is difficult to install since it does not provide a single executable module and must be compiled on the host computer.

d. Awareness Summary

This section covers three network-monitoring solutions available in the commercial and open-source domains. These network-monitoring tools are fully capable of providing situation awareness to the network administrator. Additionally, many of these solutions provide integrated mapping capabilities, though these maps are not provided in an open, distributable format. We surveyed these systems and provided a brief summary in Table 1. SolarWinds provided the easiest solution to implement and OpenNMS provided the most cost effective. The shortcoming for these complex network-monitoring tools is that they are not tailored for the tactical environment. They are heavyweight solutions that require robust servers with database access; thus, they do not meet the lightweight needs of a tactical network administrator. Secondly, none of these systems are designed to integrate with virtualization software for the purposes of training, though some integrate for the purposes of monitoring virtual devices. In general, network-monitoring systems provide more capability than required for maintaining network awareness in tactical environments and do not meet the intent of MAVNATT. However, the foundation of these solutions and the open-source code can provide a direction that the MAVNATT awareness tool can utilize in its design.

Table 1. Awareness Tools Summary

	SolarWinds NPM	Nagios	OpenNMS
Host OS Platform	Windows Server	Unix/Linux	All Major OSes
Cost	$2795 (100)	Free (Paid Available)	Free
GUI	Yes, Web-based	Yes, CGI Web-based	Yes, Web-based
Mapping	Yes (ICMP, SNMP, WMI)	Yes (Plugin Dependent)	Yes (ICMP, SNMP, WMI, LDAP, SSH)

3. Virtualization

Virtualization tools provide the ability to create an image of network devices. These virtualized devices can then be used for evaluating the effects of device changes on the entire virtualized network and, by extension, the actual network. In the context of this thesis, virtualization is central to providing a training environment for the network administrator.

It is important to understand the differences between virtualization tools in order to better understand the virtualization solutions for MAVNATT. There are two main types of virtualization solutions: type-I and type-II hypervisors [29].

Type-I hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems. For this reason, they are sometimes called bare metal or native hypervisors; they provide full virtualization [29]. VMware ESXi, Citrix Xen Server, and Microsoft Hyper-V are examples of type-I hypervisors. These solutions are "headless," requiring another computer to serve as a front-end and allow administrator interaction with the hypervisor to install, operate, and maintain the virtual machines. Type-I hypervisors do not provide any user space for applications. Therefore, in order to utilize an application like MAVNATT, we would have to install and run the application on a virtualized machine within the hypervisor. These two problems provide a level of complication that does not support the lightweight solution that MAVNATT is intended to be.

Type-II hypervisors run on a conventional operating system just as any other user application. Type-II hypervisors are called hosted hypervisors and provide para-virtualization, meaning they abstract guest operating systems from the host operating system, but the guest knows it is running in a virtualized environment [29]. VMware Fusion, Kernel-based Virtual Machine (KVM), and VirtualBox are examples of type-II hypervisors.

An extension of type-II hypervisors is the container-based virtualization that is currently found on Linux operating systems. The host operating system running a kernel-based hypervisor solution shares its architecture and kernel version to the guest operating systems to provide the guest its own process and network space instead of creating a full-fledged virtual machine. These containers do not have the overhead of a regular type-II hypervisor and are therefore extremely fast and efficient. The disadvantage is that the guest operating systems can only be of the same kernel-base has the host. Windows, Mac OS X, and other non-Linux operating systems cannot run in a container-based solution. Linux Containers (LXC), User Mode Linux (UML), and OpenVZ are examples of these container-based solutions.

From these descriptions we highlight the following observations:

- Type-I hypervisors require additional frontend resources and increased complexity to integrate the different network elements.
- Type-II hypervisors require many resources on the host system but provide a lighter-weight solution than type-I hypervisors.
- Container-based solutions are the most efficient and the lightest-weight solution but they cannot run Microsoft Windows because container guest OSs must utilize the same kernel version.

As type-II hypervisors are the most lightweight virtualization solution available and Microsoft Windows is the primary operating system family on DOD networks, this section will survey leading type-II hypervisors capable of hosting Microsoft Windows operating systems.

a. VirtualBox

VirtualBox is a type-II hypervisor that is capable of running on both 32-bit and 64-bit Windows, Linux, and Mac host operating systems computers. It can run both 32-bit and 64-bit Windows and Linux guest computers. It is compatible with many virtual machine formats. VirtualBox is written in C++ and is very well documented. It provides a comprehensive software development kit (SDK), which allows for integrating every aspect of VirtualBox with other software systems written in Java, Python, C, C++ and other languages [31], [32]. Figure 11 provides a sample image of the VirtualBox interface.

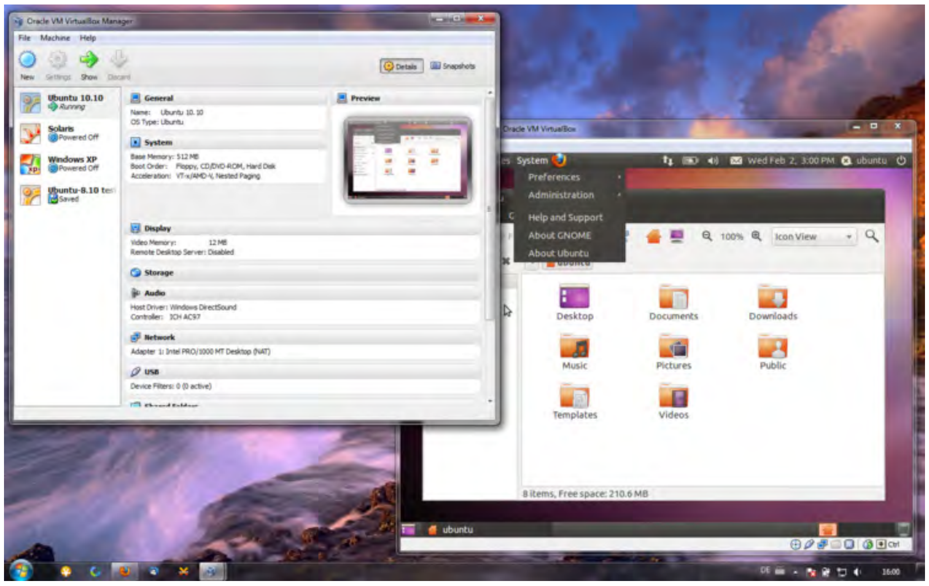


Figure 11. VirtualBox Interface on Windows, from [30]

b. VMware Fusion

VMware Fusion 7 is a type-II hypervisor that is capable of running on both 32-bit and 64-bit Windows, Linux, and Mac host operating systems. It can also run both 32-bit and 64-bit guest virtual machines of Windows, Linux, and Mac operating systems, as well as many other operating systems. VMware Fusion is compatible with the VMware VIX API, which allows integration of C-based applications [33], [34]. Figure 12 illustrates the VMware Fusion 7 interface.

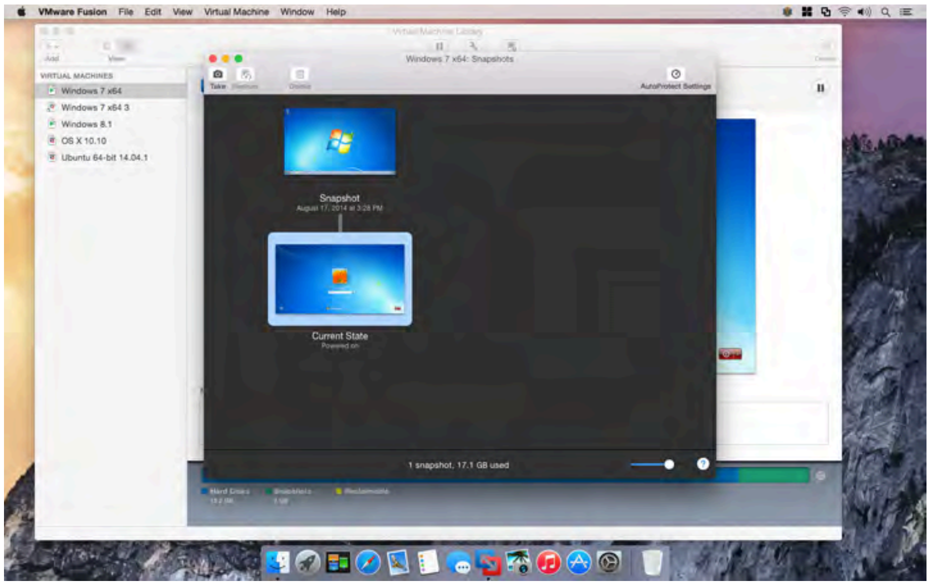


Figure 12. VMware Fusion 7 Interface, from [35]

c. Kernel-Based Virtual Machine (KVM)

Kernel-Based Virtual Machine (KVM) is a type-II hypervisor that is capable of running on both 32-bit and 64-bit Linux host operating systems. It can run both 32-bit and 64-bit virtual machines of Windows and Linux guest operating systems. KVM is written in C and utilizes a robust virtualization API, libvirt, which is also written in C [36]. Figure 13 provides a sample image of the KVM interface.

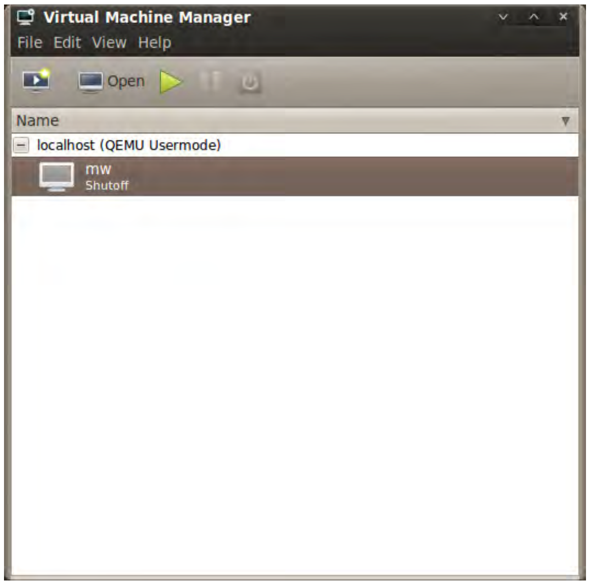


Figure 13. KVM Interface

d. Virtualization Summary

This section discusses the differences between type-I, type-II, and container-based hypervisors. As the intent of MAVNATT is to provide a lightweight tool to the tactical network administrators, type-II hypervisor-based tools are most appropriate. We surveyed three industry-leading type-II hypervisors -- VirtualBox, VMware Fusion, and KVM -- based on the requirements to be able to run Microsoft Windows, the most widely used operating system on DOD networks. All these hypervisors have APIs that support integration with other applications. Table 2 summarizes the virtualization solutions.

Table 2. Virtualization Tools Summary

	VirtualBox	VMware	KVM
Host OS Platform	All Primary OSes	All Primary OSes	Linux
Guest OS Platforms	All Primary OSes	All Primary OSes	Windows/Linux
Cost	Free	$70 (one-time fee)	Free
API Integration	Yes	Yes	Yes

4. Network Simulators

A network simulator is a type of application that models a computer network and predicts its behavior. This research will review network simulators because they closely relate to two MAVNATT functional areas -- mapping and virtualization. Network simulators can use both GUI and command line interfaces; our focus is on GUI-based solutions because they provide a level of awareness on the network topology and status. Network simulators also capitalize on virtualization techniques identified in the previous section.

a. Graphical Network Simulator (GNS3)

Graphical Network Simulator (GNS3) is an open-source graphical network simulator that was created to support training for Cisco and Juniper routers, as well as open-source router software running in virtual machines. GNS3 integrates with many type-I and type-II hypervisors that provide virtual network devices for its network simulation. GNS3 provides a variety of prepared open-source virtual appliances, and users can create their own. GNS3 is written in Python [37]. Figure 14 demonstrates the GNS3 interface.

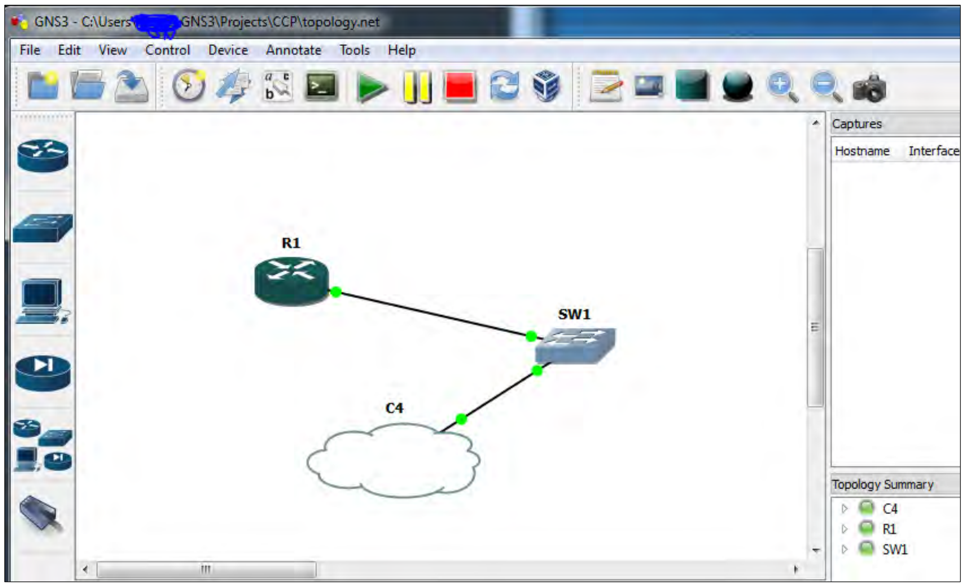

GNS3 - C:\Users\[redacted]GNS3\Projects\CCP\topology.net
File Edit View Control Device Annotate Tools Help
R1
SW1
C4
Captures
Hostname Interface
Topology Summary
C4
R1
SW1

Figure 14. GNS3 Interface, from [37]

b. *Common Open Research Emulator*

Common Open Research Emulator (CORE) is also an open-source graphical network simulator. The Naval Research Lab Networks and Communication Branch maintain CORE. It is built upon Linux as a host platform and uses the network namespace functionality in Linux containers (LXC) as a virtualization technology. This allows CORE to start up a large number of virtual machines quickly. CORE supports the simulation of fixed and mobile networks. CORE is open-source, written in Python, and it has a well-documented API [38], [39]. Figure 15 depicts the CORE interface.

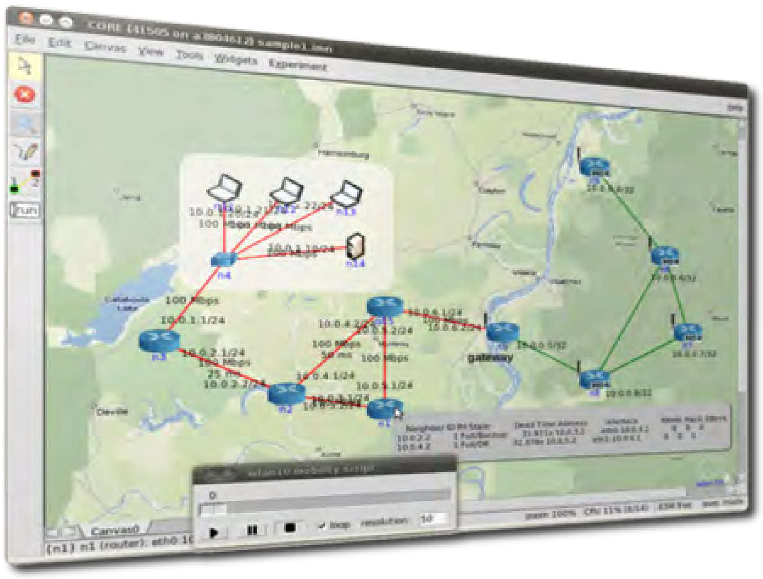
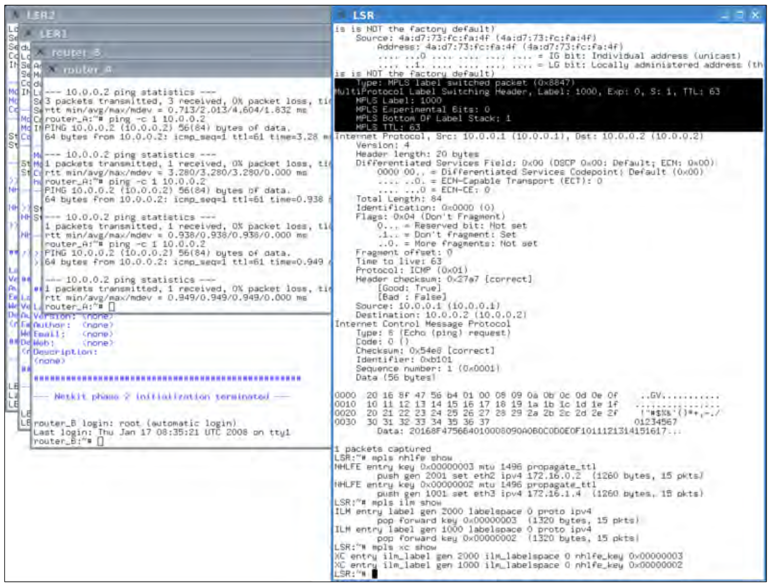


Figure 15. CORE Interface, from [38]

c. *Netkit*

Netkit is a command-line based simulation tool that uses user-mode Linux (UML) to create the virtual machines. Since the host system uses UML, all client virtual systems must be Linux-based with the same Linux kernel as the host, meaning it cannot run Windows-based virtual machines. Netkit utilizes pre-formatted labs to automatically generate the virtual machines and connections. There is a GUI plugin that can visualize these labs. The application has not been updated since 2011, but using the application demonstrates a very capable concept of employment. Netkit is open-source, written in C shell script, and has third-party APIs that generate Netkit labs [40]. Figure 16 shows the Netkit interface.



```
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, ti
rtt min/avg/max/mdev = 0.713/2.013/4.604/1.832 ms
router_A:~# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=61 time=3.28 m

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, ti
rtt min/avg/max/mdev = 3.280/3.280/3.280/0.000 ms
router_A:~# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=61 time=0.938

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, ti
rtt min/avg/max/mdev = 0.938/0.938/0.938/0.000 ms
router_A:~# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=61 time=0.949

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, ti
rtt min/avg/max/mdev = 0.949/0.949/0.949/0.000 ms
router_A:~# 
```

```
Version:     <none>
Author:      <none>
Email:       <none>
Web:         <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---

router_B login: root (automatic login)
Last login: Thu Jan 17 08:35:21 UTC 2008 on tty1
router_B:~# 
```

```
is is NOT the factory default)
    Source: 4a:d7:73:fc:fa:4f (4a:d7:73:fc:fa:4f)
        Address: 4a:d7:73:fc:fa:4f (4a:d7:73:fc:fa:4f)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
        .... ..1. .... .... .... .... = LG bit: Locally administered address (th
is is NOT the factory default)
    Type: MPLS label switched packet (0x8847)
MultiProtocol Label Switching Header, Label: 1000, Exp: 0, S: 1, TTL: 63
    MPLS Label: 1000
    MPLS Experimental Bits: 0
    MPLS Bottom Of Label Stack: 1
    MPLS TTL: 63
Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 63
    Protocol: ICMP (0x01)
    Header checksum: 0x27a7 [correct]
        [Good: True]
        [Bad : False]
    Source: 10.0.0.1 (10.0.0.1)
    Destination: 10.0.0.2 (10.0.0.2)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0 ()
    Checksum: 0x54e8 [correct]
    Identifier: 0xb101
    Sequence number: 1 (0x0001)
    Data (56 bytes)

0000  20 16 8f 47 56 b4 01 00 08 09 0a 0b 0c 0d 0e 0f    ..GV...........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                           01234567
        Data: 20168F4756B40100080900A0B0C0D0E0F1011121314151617...

1 packets captured
LSR:~# mpls nhlfe show
NHLFE entry key 0x00000003 mtu 1496 propagate_ttl
        push gen 2001 set eth2 ipv4 172.16.0.2  (1260 bytes, 15 pkts)
NHLFE entry key 0x00000002 mtu 1496 propagate_ttl
        push gen 1001 set eth3 ipv4 172.16.1.4  (1260 bytes, 15 pkts)
LSR:~# mpls ilm show
ILM entry label gen 2000 labelspace 0 proto ipv4
        pop forward key 0x00000003  (1320 bytes, 15 pkts)
ILM entry label gen 1000 labelspace 0 proto ipv4
        pop forward key 0x00000002  (1320 bytes, 15 pkts)
LSR:~# mpls xc show
XC entry ilm_label gen 2000 ilm_labelspace 0 nhlfe_key 0x00000003
XC entry ilm_label gen 1000 ilm_labelspace 0 nhlfe_key 0x00000002
LSR:~# 
```

Figure 16. Netkit Interface, from [40]

d. Network Simulator Summary

Network simulators demonstrate their utility as training and evaluation tools. Netkit is interesting due to its ability to use third-party applications to automatically generate labs and virtualized networks, but it lacks developer support, relies on UML that does not support Microsoft Windows, and has limited API support. CORE has a clean interface and is easy to use, but its reliance on LXC means it also lacks the ability to virtualize Microsoft Windows. GNS3 is the dominant solution and is a proven industry leader in network simulation. Table 3 summarizes the Network Simulation tools.

Table 3. Network Simulation Tools Summary

	GNS3	CORE	Netkit
Host OS Platform	All Primary OSes	Linux	Linux
Virtualization Host	VirtualBox, Qemu	LXC	UML
Virtualization Guest	All Primary OSes	Linux	Linux
Virtual Routers	Cisco, Juniper, Quagga, Linux-Based	Quagga, Linux-Based	Quagga, Linux-Based
Cost	Free	Free	Free
API	Yes	Yes	Yes, 3rd Party

5. Current Tools Survey Summary

The purpose of this survey was to identify capabilities that are critical to tactical level network operations. We discovered that the overall functionality that is critical to tactical network operations could be collapsed into two major application areas: network monitoring and network simulators.

Network monitoring tools, like SolarWinds, provide an excellent visualization tool that improves network administrator situational awareness and understanding of the network. These tools poll the network at set intervals to provide fault, performance, and accounting status. Network monitoring suites also have integrated mapping tools that automatically discover the network. They conduct both monitoring and discovery through multiple network management protocols like ICMP, SNMP, and WMI. The survey discovered that network-monitoring tools are not utilized for network training and evaluation, and they do not integrate into virtualization hypervisors and tools. Without the capability to conduct network training in a virtual environment, network-monitoring tools cannot be used as a single-source solution for MAVNATT.

Network simulation tools, like GNS3, provides an excellent visualization tool, as well as an integrated virtualization platform that improves network administrators a capability to be trained and evaluated on networking equipment. All network administrator-training events could also be conducted on a network simulation tool. Generally, network simulators do not have import and auto-generation capabilities. The only exception was Netkit. Therefore, network simulations are manually generated in the GUI and linked to the virtual or emulated devices. Network simulators can be integrated into a live network, but they are not designed to overlay live and virtual components that represent the same terminal device. Network simulators are also unable to provide situational awareness to the network administrator. The lack of integrated network awareness capabilities, the lack of auto-generation of network components in virtualization solutions, and the inability to overlay live and virtual components within the GUI demonstrates that network simulators are unable to be used as a single-source solution for MAVNATT.

Commercial and academic solutions were surveyed and found to be unable to singularly meet the identified requirements to support tactical level network operations. Figure 17 identifies the gap between network monitoring tools and network simulation tools. This is the gap that MAVNATT is intended to fill, covering both the capabilities of network monitoring tools and network simulation tools. The next chapter outlines a framework and methodology to integrate those capabilities identified during this survey of commercial solutions.

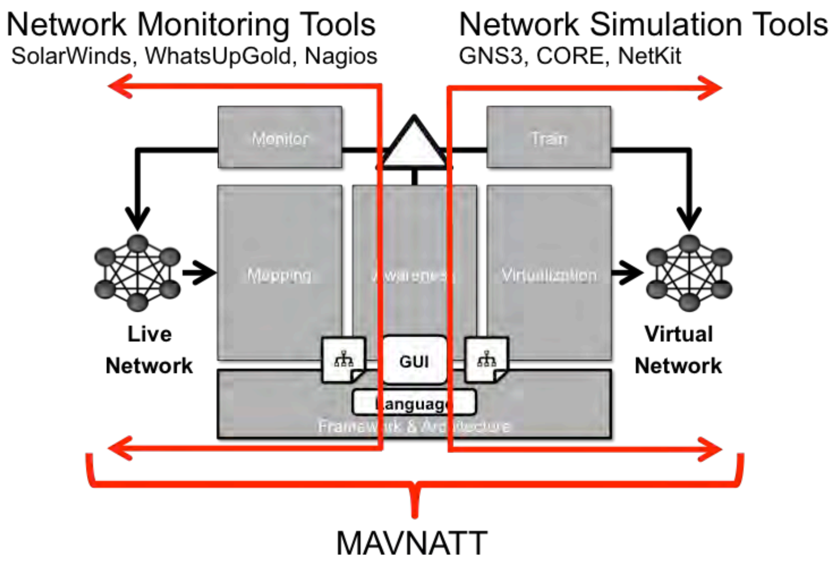


Figure 17. MAVNATT Model compared to Network Monitoring and Network Simulation Tools

D. SUMMARY

This chapter presented training requirements for military network administrators, the current tools in industry and academia, and proposed tools to be utilized as the framework for MAVNATT. The ability to train network administrators in operational environments is lacking and a serious gap exists at the tactical level. In order to fill this gap, we reviewed commercial and academic solutions. The review identified that network monitoring suites and network simulators provide the functionality desired in the MAVNATT project, but there is not a system that combines them into a single lightweight solution.

In the next chapter, we design a framework and architecture based on the tools surveyed in order to integrate the MAVNATT functional areas: mapping, awareness, and virtualization.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN AND METHODOLOGY

A. OVERVIEW

The previous chapter discusses commercial solutions for network administrators and it identifies that they do not meet requirements of a lightweight system that supports operations and training in a tactical environment. With this in mind, we have developed the MAVNATT conceptual model, depicted in Figure 18.

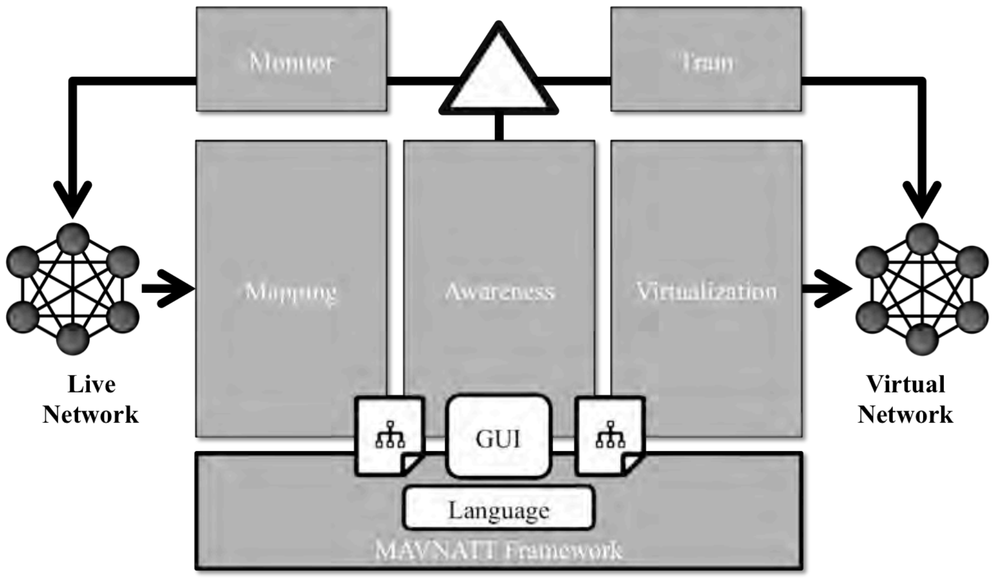


Figure 18. MAVNATT Conceptual Model

In this chapter, we detail outline our intentions for the MAVNATT architecture and framework. We then identify the design for the framework components, review available solutions, and provide component level prototypes to demonstrate desired capabilities.

B. MAVNATT ARCHITECTURE

Many network administration tools exist that support both operations and training, but not in an integrated application. Network monitoring tools and network simulation tools are the application types that best meet the requirements outlined in the conceptual model. Network monitoring tools provide awareness and mapping capabilities while network simulation tools provide awareness and virtualization capabilities. The combined capabilities of these tools align to the previously defined functional areas: mapping, awareness, and virtualization.

The MAVNATT architecture builds its module base from these functional areas to meet the operational and training needs for tactical network administrators. This section briefly discusses the objectives and requirements for each module within the MAVNATT architecture.

1. Mapping

Mapping provides a network administrator the capability to understand the physical layout, composition, and interconnectivity of the network.

a. Objectives

- The main objective of the mapping module is to increase the network administrator's situational awareness by providing a complete and accurate view of the physical topology of the network in order to maintain availability and security of the network.
- The secondary objective of the mapping module is to support network management by being able to verify a physical network topology against the planned network topology, allowing the network administrator to understand the effect of network changes.

b. Requirements

- The module must communicate with the awareness module. Specifically, it must be able to accept arguments, including an existing network topology. It must return a full or incremental network topology.
- The arguments should include, but are not limited to, elevated privileges, scan types such as full or incremental, and scan modes such as passive or active.

2. Awareness

Awareness provides a visual representation of the network and a status of network devices; it also provides a platform to visualize training scenarios for the network administrator.

a. Objectives

- The main objective of the awareness module is to enhance a network administrator's situational awareness by providing a visualization platform to display the network topology and status of network devices.
- The secondary objective of the awareness module is to provide a capability to overlay virtualized devices to support network administrator training and evaluation.

b. Requirements

- The module must be able to import a network topology and display it.
- The module must be able to manually generate a network topology and display it.
- The module must provide network device availability status, at minimum.
- The module must communicate with the mapping module. Specifically, it must be able to call the module with appropriate arguments to get a full or incremental network topology, and the module must be able to display it.
- The module must communicate with the virtualization module. Specifically, it must be able to provide the virtualization module with a description of a network topology to be virtualized. It should accept a set of information that allows the awareness module to overlay and integrate virtual devices to support training and evaluations.

3. Virtualization

Virtualization provides a completely partitioned environment that allows network administrators to train and be evaluated without interfering with the operational network.

- Objectives
- The main objective of the virtualization module is to integrate MAVNATT with a hypervisor that can support the virtualization of a network topology that is logically partitioned from the operational network.

a. Requirements

- The module must communicate with the awareness module. Specifically, it should receive a network topology that is to be virtualized and it should return a set of connectors to those virtual devices.
- The module must ensure the virtual devices are unable to interfere with the operational network.
- The module must infer a virtual device whenever a full specification is not available.

4. Architecture Summary

This section briefly covered each of the modules in the MAVNATT architecture and described how they are expected to operate as a module within the principal system. This modularity within the architecture allows for a systematic approach to the design and development of the MAVNATT system.

C. MAVNATT FRAMEWORK

The MAVNATT framework identifies the components necessary to integrate the three modules that compose the MAVNATT architecture. Analysis of the objectives and requirements of the MAVNATT architecture modules allow us to determine the components of the MAVNATT framework by identifying elements of the modules that are dependent on external modules or users. These framework components are important to providing consistency for the data exchanged between modules, input and output to the user, and usability of the system. The components are identified as follows: network topology format, programming language, GUI, and virtualization platform API. Figure 19 visually represents the framework and its components.

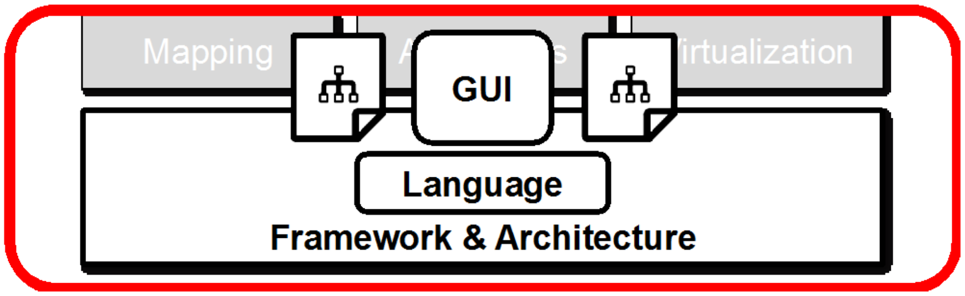


Figure 19. MAVNATT Framework

The framework provides the integration points between the three architecture modules, which align to our functional areas. The following sections detail each framework component as well as develop a component level prototype to demonstrate the desired capability.

D. NETWORK TOPOLOGY FORMAT

MAVNATT requires a file format component capable of describing the network topology. This format is used to represent the topology among the three MAVNATT modules. Figure 20 depicts the integration points of the network topology format in relation to the functional areas. The mapping functional area can discover the network and save it to this format in order to pass it to the awareness module. The awareness functional area can use it to store information about the status of the network it is monitoring. The virtualization functional area can use it to describe the network to be virtualized for training or evaluation. This format also provides network administrators the ability to easily create, save, export, and exchange a network topology up and down the military hierarchy. A common network topology format simplifies prototype development and allows MAVNATT to maintain modularity. This section details objectives, reviews three formats considered for this component, and establishes a component prototype.

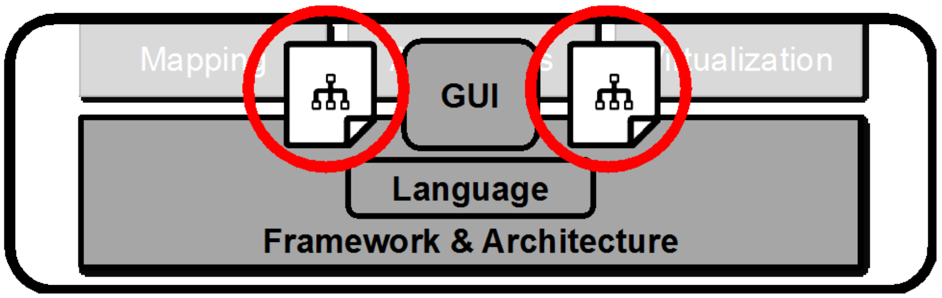


Figure 20. MAVNATT Network Topology Format Integration

1. Network Topology Format Objectives

- The network topology format must be human readable in order to support the manual interpretation of a network topology, integration into network operations planning documents, and to facilitate the education and training of network administrators.
- Implementations must provide the capability of reading the network topology format into a usable representative object that allows for the addition and deletion of topology information.
- Implementations must provide the capability of writing the representative object to file in the network topology format.
- The topology format should be extendable in order to support the recognition of all current and future network topology devices including: computers, switches, routers, VOIP, printers, mobile devices, etc.
- The topology format should allow for full and partial network representation. This allows network administrators to receive and interpret a full network topology from the mapping module and it allows the virtualization module to receive and interpret a small portion of the network to be virtualized, which reduces resource requirements for training scenarios.

2. Network Topology Format Review

We review three graph formats as candidates for the network topology framework component based on industry and research standards: Neo4j, JavaScript Object Notation (JSON), and GraphML.

a. Neo4j

Neo4j is popular in industry and it is currently being utilized by MARFORCYBER to support the cyber-centric common operational picture. Neo4j is a graph database that utilizes a Java Virtual Machine based NOSQL database server to store data structure as graphs rather than tables. Neo4j is available in a free community version and a paid enterprise version. Neo4j's strengths lie in its relational capabilities. It can derive relations in larger amounts of data. It is a scalable and dynamic environment with very capable APIs for all principal programming languages. It also natively imports many other graph formats, including JSON and GraphML. However, Neo4j's strength is also its principal shortfall for MAVNATT. MAVNATT is intended to be a lightweight system. Adding additional overhead in the form of a database server complicates the operating requirements for the network administrators and it increases the complexity in developing and delivering a prototype. For these reasons, we will not pursue Neo4j further in this research. Nonetheless, Neo4j may warrant investigation after the initial prototype is established [41], [42].

b. JSON Format

JavaScript Object Notation, JSON, is an open Internet standard for a lightweight, text-based, language-independent data interchange format. JSON defines a small set of formatting rules for the portable representation of structured data. Its design makes it very fast compared to XML formats at referencing and exchanging data, but it is also very resource intensive on the host system. JSON's main shortfall as a candidate for MAVNATT is that it is not specifically utilized to describe a network or graph; it only describes the way data is stored and exchanged. For this reason, JSON could improperly tag data that is specifically meant to describe a network topology [43], [44].

c. GraphML Format

GraphML is an XML-based file format for graphs that comprehensively describes the structural properties of a graph and has simple attribute extension mechanisms. As it is XML-based, it is inherently an open Internet standard. GraphML resulted from the joint effort of the graph drawing community to define a common format for exchanging graph-structure data. GraphML provides a strict definition of graph elements to represent the network topology [44], [45].

d. Network Topology Format Survey Summary

Neo4j provides the most innovative solution for a topology format, but it is a heavyweight solution requiring a database backend. We choose not to pursue Neo4j, at this point, because it does not meet the lightweight requirements for the MAVNATT system. JSON and GraphML are more simplistic and lightweight solutions that will allow for faster prototyping and deployment.

3. Network Topology Format Component Prototypes

The JSON and GraphML prototypes will both be generated in the following environment:

- OS: Mac OS X 10.10.3
- IDE: NetBeans IDE 8.0.2 with the community Python Plugin
- Python: Version 3.4
- NetworkX: Version 1.9.1 - NetworkX is an open-source package developed by the Los Alamos National Laboratory for the study of graphs. In our prototype, it is used to allow us to import and export JSON data, read and write GraphML files, and it generates a graph object that represents nodes and edges.
- JSON File: A simple JSON file with nodes and edges in the proper format to be imported by NetworkX.
- GraphML File: A simple GraphML file with nodes and edges in the proper format to be imported by NetworkX.

a. JSON Prototype

NetworkX does not read the JSON directly from file. Instead, the JSON file must be loaded into a string and then NetworkX imports the string into a NetworkX graph object. Once the graph object is created, the JSON format is no longer required. The graph object consists of nodes, edges, and attributes. These are easily modified, added, and removed with NetworkX methods. The graph object can then be saved in JSON format by converting the NetworkX graph object to a string and saving it to file. The prototype is capable of reading and writing JSON formats.

b. GraphML Prototype

NetworkX natively reads GraphML from file into a NetworkX graph object. The NetworkX graph object is used for storing edge and node information, just like the JSON implementation. NetworkX can then natively save the graph object to GraphML file format. The prototype is capable of reading and writing JSON formats.

4. Network Topology Format Component Summary

The network topology format as a component of the MAVNATT framework provides the ability to communicate information among all three of the MAVNATT modules. The JSON and GraphML prototypes successfully read and write the topology. Both prototypes also demonstrated extendibility by allowing for the addition, deletion, and modification of topology edges and nodes. It is also extendable by allowing additional attributes to be added to the data structure. Both formats are human readable, though GraphML is slightly better because the format allows more information to be presented to the reader. In conclusion, both JSON and GraphML meet the network topology format requirements and successful prototyping.

E. PROGRAMMING LANGUAGE

MAVNATT requires a programming language that standardizes the libraries, packages, and APIs among all three MAVNATT modules. A common programming language also allows for rapid development of framework components and architecture modules. Figure 21 depicts that the programming language supports all the MAVNATT components and the overall architecture.

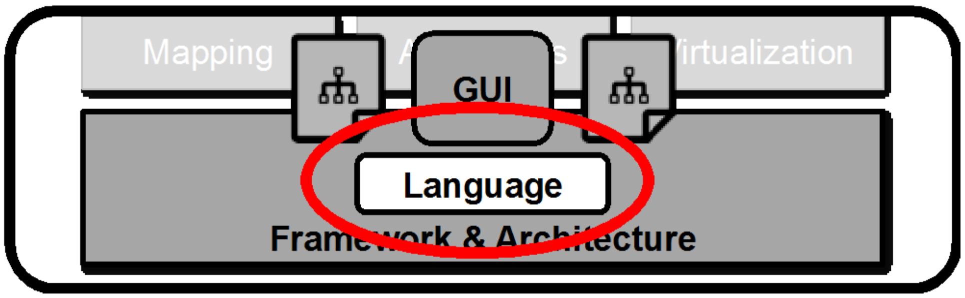


Figure 21. MAVNATT Programming Language Integration

This section will detail programming language objectives and review three programming languages considered for this component. The programming language component is not a standalone component but rather is integral to, and common among, the other components. Therefore, it will not be individually prototyped, but it will be used in other component prototypes.

1. Programming Language Objectives

- The programming language must be capable of implementing the libraries and APIs required for other framework and module purposes, including: graph formats, GUI, and virtualization APIs.
- The programming language must be portable, supporting a "write once, run anywhere" methodology to be run on multiple platforms, including Windows, Linux, and Mac.
- The programming language must support rapid development for the purposes of generating component and modular prototypes in support of the overarching MAVNATT project.

2. Programming Language Review

We review three programming languages as candidates for the framework component based on industry standards and author familiarity: C++, Java, and Python.

a. C++

C++ allows for low-level memory manipulation and is the core of many modern object oriented languages. It excels at performance-critical applications. However, C++ strengths do not outweigh its weaknesses in the area of rapid prototyping. It has been demonstrated that C++ code takes longer to produce, is more error prone, and its improper use can introduce more security risks to the system than the other languages. Further, it must be explicitly compiled for each target machine implementation. C++ will not be pursued MAVNATT. [46]-[48]

b. Java

Java is a high level, interpreted, and object-oriented programming language that is capable of being compiled into packaged byte-code that can execute on any system with a Java Virtual Machine (JVM) regardless of physical computer architecture, making it platform independent. The JVM provides a single target platform that simplifies prototyping and deployment of software. Since it abstracts away some of the complexity of C/C++, the languages upon which Java was built upon, it allows programmers to increase their productivity. Java does have an integrated graphical user interface API and a large number of available packages. Research into the object-oriented class structures, ease of programming, and error reduction demonstrates that Java is preferable to C++ [46]–[48].

c. Python

Similar to Java, Python is a high level, interpreted and object-oriented programming language. Python applications can be easily distributed and run on any system with a Python interpreter. It abstracts away from the programmer much of the complexity of C++ and Java, allowing the programmer to be more productive by quickly, precisely, and accurately creating applications. Python is also object-oriented and has a strong standard library, including an integrated graphical user interface API. Research further demonstrated that Python was among the top languages for the most error-free and most rapidly producible code [46]-[48].

d. Programming Language Review Summary

C++ is widely recognized for its speed and access to low-level memory, but it does not support rapid development and platform portability. Java and Python both support platform portability, rapid development, and a wide range of available APIs to support requirements. Java is more widely deployable than Python because the application can run on any platform with the Java Virtual Machine, whereas Python is more closely tied to the OS implementation of its Python interpreter. Python is better than Java in rapid development capabilities because its use results in fewer code errors.

3. Programming Language Component Summary

The programming language as a component of the MAVNATT framework provides the ability to utilize standardized libraries, packages, and APIs among all three of the MAVNATT modules. Individual prototypes for this component were not developed, but both Python and Java are used to prototype other components required for the MAVNATT framework. In conclusion, both Java and Python provide flexibility by supporting rapid code development, platform flexibility, and support for many packages and APIs that support MAVNATT framework components and module development.

F. GUI

The MAVNATT framework requires a GUI to provide a visualization tool for the network administrator. MAVNATT must be capable of representing both operational and training environments. A GUI-based visualization tool, vice a console-based system, provides more situational awareness of the physical and virtual network topologies, network device faults, and training scenarios. GUIs simplify information necessary for network administrators to make decisions and therefore increase productivity, especially among novice users. Figure 22 demonstrates that the GUI is heavily tied to the awareness module, but it also must represent information from both the mapping and virtualization modules.

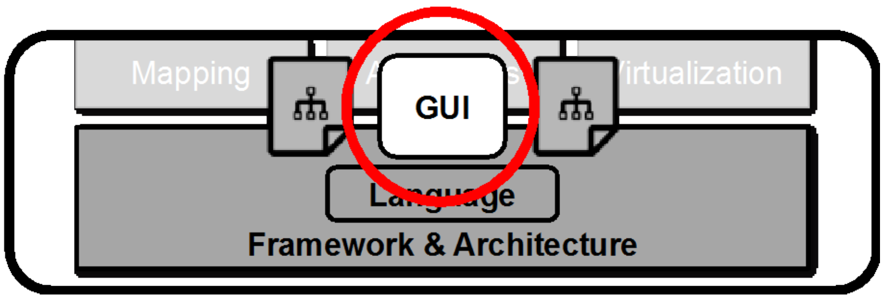


Figure 22. MAVNATT GUI Integration

This section will detail GUI objectives, survey three GUIs considered for this component, and establish a component prototype. Figure 23 provides a mockup of the projected GUI interface, which demonstrates the desired simplicity and lightweight interface.

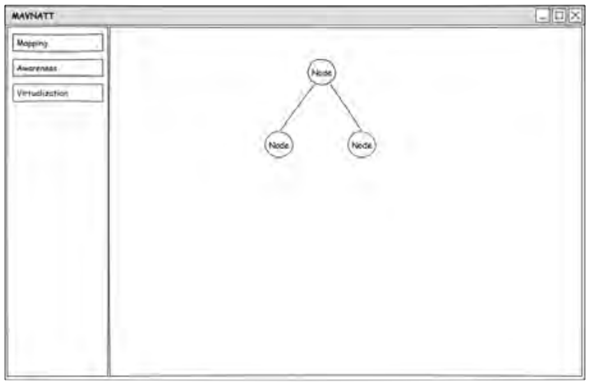


Figure 23. Mockup - MAVNATT GUI

1. GUI Objectives

- The GUI must be simple and intuitive, abstracting complexity away from the network administrator while increasing situational awareness.
- The GUI must be capable of representing the network topology format.
- The GUI must visualize a composition of both the live and virtual networks.

2. GUI Review

We survey three GUI APIs as candidates for the framework component based upon integration with our selected programming languages: Java Swing, Python Tkinter, and Python Qt.

a. Java Swing GUI

Java's Swing API provides a set of components and widgets to provide GUI functionality to Java-based applications. Swing is part of the Java Foundation Classes that form Java's graphical framework. It is deployable on all modern operating systems and can provide the look and feel of that OS or a common look and feel across all OSes [49].

b. Python Tkinter GUI

Tkinter is a Python GUI API that is part of the standard Python library. This means that any system that can run Python can run Tkinter applications. Tkinter stands for “Tk interface” which derives from the windowing toolkit of the Tool Command Library (Tcl). Tcl/Tk combined into a simple method for creating cross-platform GUI applications, which Python extended for its GUI interface. Tkinter provides the core components of a graphical user interface, but it is recognized as not being as elegant as other GUI solutions [50].

c. Qt GUI

Qt is a cross-platform GUI API that provides the ability to create commercial grade interfaces. Qt is more complex than Tkinter, but it also has more widget capabilities. Third party applications have been designed to generate Qt GUIs that can be imported into applications. Qt has two licensing methods, LGPL and Commercial, which induces some complexity into the direction MAVNATT could go in future prototypes [51].

d. GUI Review Summary

The GUI framework component is essential to the performance and success of the MAVNATT system. Swing and Tkinter provide integrated solutions for prototypes developed in Java and Python without the requirement for external libraries. Qt provides a more modern and clean interface over Tkinter, but it has licensing requirements and relies on installation of external libraries. Qt will not be pursued for prototyping.

3. GUI Component Prototype

The GUI component prototype will be developed in the following environment:

1. OS: Mac OS X 10.10.3
2. IDE: NetBeans IDE 8.0.2 with the community Python Plugin
3. Python: Version 3.4 with Tkinter

a. Python Tkinter GUI Prototype

The Python Tkinter GUI prototype meets all designated component objectives. It provided a simple, lightweight, and intuitive interface that allowed for efficient depiction of a network topology. Figure 24 demonstrates the Python Tkinter GUI prototype.

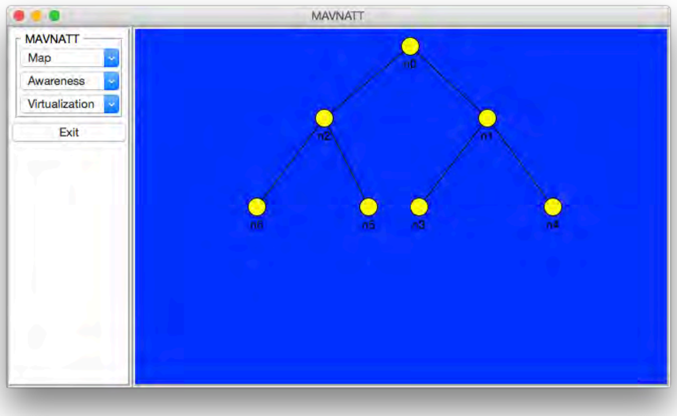


Figure 24. MAVNATT GUI Component Prototype

It was easy to generate widgets within the GUI that could be clicked and moved. Each item on the GUI canvas is issued an ID that can be used to change the attributes of that item including size and color. Shapes are not easy to change and would require deleting an item and adding a new one with a new ID. Compositing a layer that represents the virtual network topology for network administrator training and evaluation would be possible because of this.

4. GUI Component Summary

The GUI framework component is essential to the visualization of the network topology and providing the network administrator the ability to interact with the application. The GUI component prototype successfully established a simple and intuitive interface that represented a network topology. The interface is not as refined as production level applications, but the prototype demonstrated that built-in GUI capabilities of programming languages are sufficient. In conclusion, Python Tkinter meets the GUI component requirements and successful prototyping.

G. VIRTUALIZATION API

The most important component of MAVNATT is its integration into a virtualization platform. Without integrated virtualization, MAVNATT fails to provide a training platform to network administrators and does not set itself apart from a simple network management suite. Virtualization also allows the architecture to overlay the physical topology without interfering with network operations. Figure 25 demonstrates that the relation between the awareness and virtualization modules.

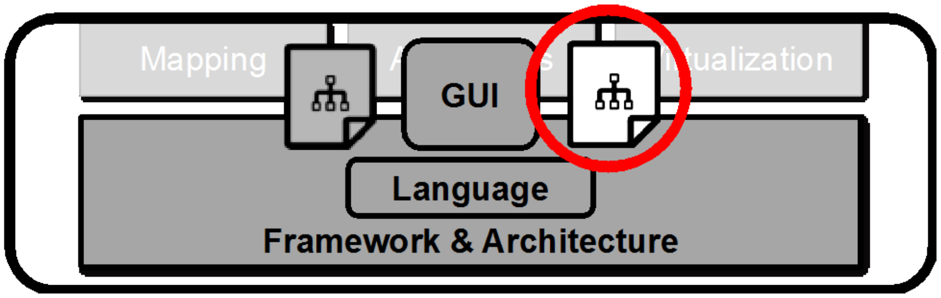


Figure 25. MAVNATT Virtualization API Integration

This section details the virtualization API objectives and establishes a component prototype. In Chapter II, we discussed VirtualBox and VMware and identified them as the most capable type-II hypervisor solutions for MAVNATT. For the purposes of our research and streamlining prototyping, we will use VirtualBox and its API because it is free, open source, and it has a comprehensive SDK that can be bound to both Java and Python programming languages [31], [32].

1. Virtualization API Objectives

- The virtualization API must automate all interaction with the hypervisor.
- The virtualizing API must allow for the accessing of all virtual machine (VM) information from the hypervisor (i.e., name, network, hardware information, etc.).

- The virtualization API must allow for cloning individual systems. This is critical to support rapid deployment of virtual networks in a tactical environment.

2. Component Prototypes

The Java VirtualBox API and Python VirtualBox API prototypes will both be generated in the following environment:

- OS: Mac OS X 10.10.3
- IDE: NetBeans IDE 8.0.2 with the community Python Plugin
- Python: Version 2.7
- Java: Version 8 update 45
- VirtualBox Type-II Hypervisor: Version 4.3.26 r98988
- VirtualBox: Version 4.3

a. Java VirtualBox API

The Java VirtualBox API allows for two modes of operation, a simple object access protocol (SOAP) implementation and a cross platform component object model (XPCOM) implementation. In short, the SOAP implementation is a web-based remote procedure call. Whenever VirtualBox is installed, a web service runs in the background that accepts commands and executes them in the VirtualBox application. Conversely, XPCOM sends those commands directly to the VirtualBox application. Therefore, the XPCOM is much faster than SOAP and reduces overhead. After the package was properly installed, it The Java VirtualBox API prototype successfully meets our initial requirements.

b. Python VirtualBox API

The Python VirtualBox API has access to SOAP and XPCOM implementations. The Python VirtualBox API natively runs XPCOM without requiring system properties to be set and without much trouble importing the package. The coding of the Python implementation was less complex than Java. The Python Version 2.7 VirtualBox API prototype successfully meets our requirements.

3. Virtualization API Component Summary

The virtualization framework component is the most critical to the success of the MAVNATT system. Virtualization of the network topology provides the capability to train and evaluate network administrators. Complex architectures will be difficult to virtualize, but the VirtualBox API supports many of the tasks required for a successful production environment. In conclusion, both the Python and Java VirtualBox API implementations meet the virtualization component requirement and successful prototyping.

H. COMPONENT PROTOTYPE SUMMARY

This section identifies the framework components required to integrate the MAVNATT modules: network topology format, programming language, GUI, and virtualization API. Furthermore, prototypes of each framework component demonstrate the requirements and desired capability. GraphML and JSON both support network topology formats, though GraphML provides more information in its human readable format. Java and Python programming languages provide robust libraries, cross platform capability, and support rapid development and prototyping. Java Swing and Python Tkinter are integrated into their respective programing languages and demonstrate the ability to provide simple and intuitive interfaces to represent a network topology. Lastly, the Java and Python VirtualBox APIs both demonstrated the ability to automatically interact with the hypervisor and clone a virtual machine. Table 4 provides a summary and comparison of the framework components.

Table 4. Component Prototype Summary

	Primary	Alternate
Topology Format	GraphML*	JSON
GUI	Python Tkinter	*Not Prototyped*
Language	Python	Java
Virtualization API	Python API	Java API

Though GraphML was successful in providing baseline data for the network topology, it is recommended in following chapters that more advanced XML-based topology representations be researched or developed to support MAVNATT.

I. SUMMARY

This chapter presented the design MAVNATT architecture and framework. The architecture aligned to the three functional areas: mapping, awareness, and virtualization. The framework identified components that are used to integrate the modules: network topology format, programming language, GUI, and virtualization API. This chapter also provided framework component prototypes that demonstrated the individual component's capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

A. OVERVIEW

The previous chapter discussed the architecture and framework proposed for MAVNATT. This chapter presents an integrated prototype that implements the architecture. The implementation describes the activity flow as it integrates the architecture and framework, and illustrates the desired capabilities.

B. INTEGRATED PROTOTYPE

An integrated prototype is required to clarify the direction taken for MAVNATT and to simulate the desired capabilities of an operational system. It is not intended to be the final solution, but builds upon the concepts of the architecture and framework by describing and demonstrating the interoperability between modules and operating them in a single environment. This section defines the prototype design and capabilities.

1. Prototype Design

The prototype uses the following environment:

- OS: Mac OS X 10.10.3
- IDE: NetBeans IDE 8.0.2 with the community Python Plugin
- Network Topology Format: GraphML Version 1.0
- Programming Language: Python Version 2.7 with Tkinter
- GUI Library: Python Tkinter from Python 2.7
- VirtualBox Type-II Hypervisor: Version 4.3.26 r98988
- VirtualBox API: Version 4.3

2. Prototype Capabilities

The capabilities we present are identified to support the implementation in the next section. The architecture modules are mostly representative and rely heavily upon the work established with the framework components.

- Mapping: The mapping module is not implemented in this prototype; instead, its capability is demonstrated by importing a GraphML file representing a network topology into the awareness module. As mentioned in Chapter 3, this module is designed to generate a GraphML file that represents the real operational network.
- Awareness: The awareness module provides a GUI interface to link the separate modules. This capability is demonstrated by displaying the imported GraphML formatted topology within the GUI. It also provides a ping-based test on nodes to see whether or not the represented nodes are online and available.
- Virtualization: In the current implementation, the virtualization module is very limited. The virtualization capability is demonstrated by using a baseline configuration of a virtual machine to generate clones for each node in the network topology.

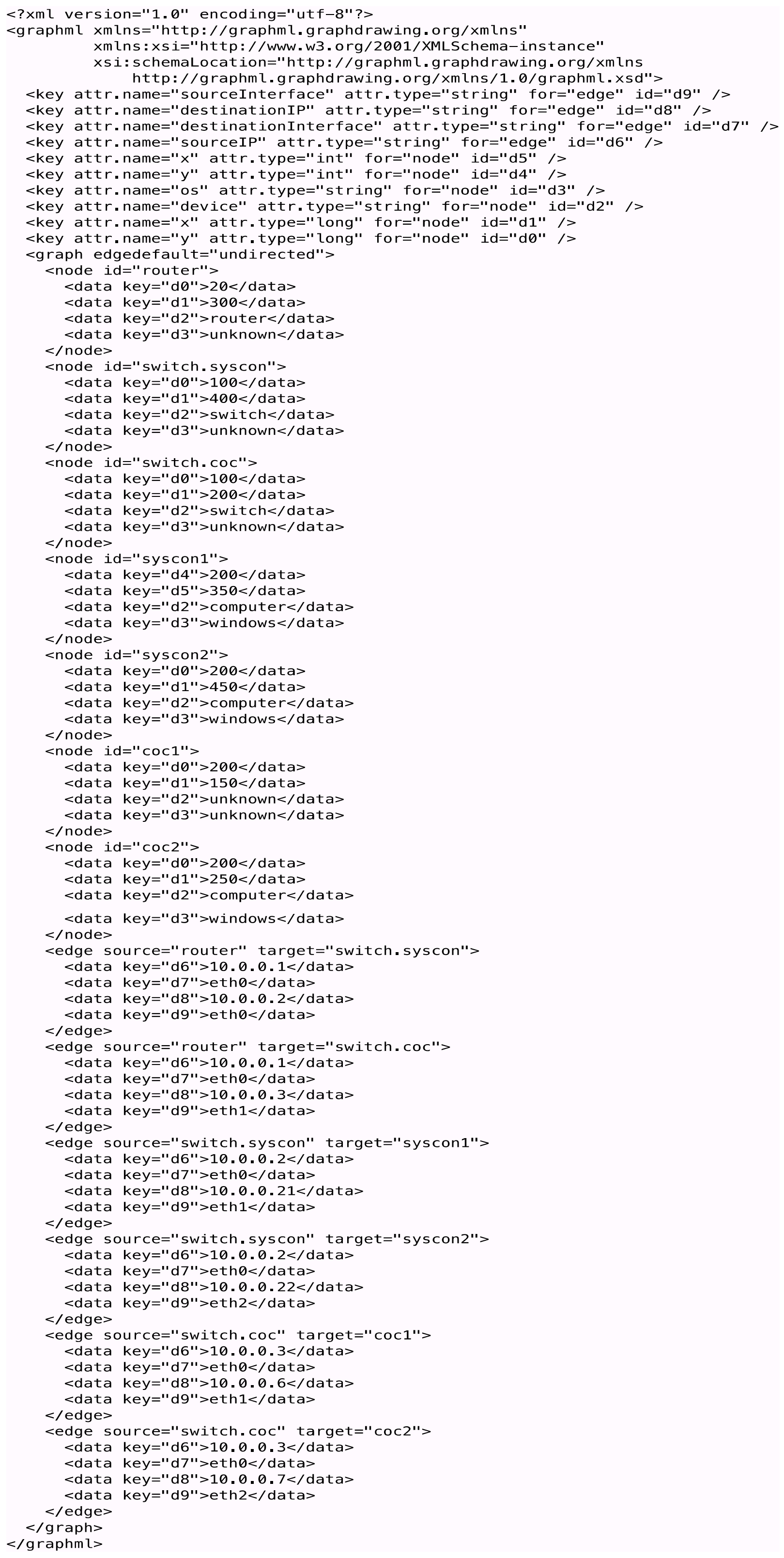
C. IMPLEMENTATION

1. Overview

This section presents a systematic flow and step-wise approach describing implementation goals desired of the final product. The integrated prototype is used to demonstrate each of the steps to the reader with respect to the current implementation as well as a discussion of the desired full-scale implementation.

2. Mapping

The mapping module is designed to discover the network topology by using active and passive methods to identify network devices and connections. It generates a GraphML file to represent the network topology that is easily saved or transferred between modules. This GraphML description of the network topology represents nodes and edges. A node or edge has attributes that are identified by key-value pairs in XML format. Attributes can be added or removed from the node or edge, as long it is a data type recognized by GraphML and it is directly related to the entity; GraphML does not support nested data structures. Figure 26 is the GraphML network topology representation for this implementation.



```xml
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
             http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key attr.name="sourceInterface" attr.type="string" for="edge" id="d9" />
  <key attr.name="destinationIP" attr.type="string" for="edge" id="d8" />
  <key attr.name="destinationInterface" attr.type="string" for="edge" id="d7" />
  <key attr.name="sourceIP" attr.type="string" for="edge" id="d6" />
  <key attr.name="x" attr.type="int" for="node" id="d5" />
  <key attr.name="y" attr.type="int" for="node" id="d4" />
  <key attr.name="os" attr.type="string" for="node" id="d3" />
  <key attr.name="device" attr.type="string" for="node" id="d2" />
  <key attr.name="x" attr.type="long" for="node" id="d1" />
  <key attr.name="y" attr.type="long" for="node" id="d0" />
  <graph edgedefault="undirected">
    <node id="router">
      <data key="d0">20</data>
      <data key="d1">300</data>
      <data key="d2">router</data>
      <data key="d3">unknown</data>
    </node>
    <node id="switch.syscon">
      <data key="d0">100</data>
      <data key="d1">400</data>
      <data key="d2">switch</data>
      <data key="d3">unknown</data>
    </node>
    <node id="switch.coc">
      <data key="d0">100</data>
      <data key="d1">200</data>
      <data key="d2">switch</data>
      <data key="d3">unknown</data>
    </node>
    <node id="syscon1">
      <data key="d4">200</data>
      <data key="d5">350</data>
      <data key="d2">computer</data>
      <data key="d3">windows</data>
    </node>
    <node id="syscon2">
      <data key="d0">200</data>
      <data key="d1">450</data>
      <data key="d2">computer</data>
      <data key="d3">windows</data>
    </node>
    <node id="coc1">
      <data key="d0">200</data>
      <data key="d1">150</data>
      <data key="d2">unknown</data>
      <data key="d3">unknown</data>
    </node>
    <node id="coc2">
      <data key="d0">200</data>
      <data key="d1">250</data>
      <data key="d2">computer</data>

      <data key="d3">windows</data>
    </node>
    <edge source="router" target="switch.syscon">
      <data key="d6">10.0.0.1</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.2</data>
      <data key="d9">eth0</data>
    </edge>
    <edge source="router" target="switch.coc">
      <data key="d6">10.0.0.1</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.3</data>
      <data key="d9">eth1</data>
    </edge>
    <edge source="switch.syscon" target="syscon1">
      <data key="d6">10.0.0.2</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.21</data>
      <data key="d9">eth1</data>
    </edge>
    <edge source="switch.syscon" target="syscon2">
      <data key="d6">10.0.0.2</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.22</data>
      <data key="d9">eth2</data>
    </edge>
    <edge source="switch.coc" target="coc1">
      <data key="d6">10.0.0.3</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.6</data>
      <data key="d9">eth1</data>
    </edge>
    <edge source="switch.coc" target="coc2">
      <data key="d6">10.0.0.3</data>
      <data key="d7">eth0</data>
      <data key="d8">10.0.0.7</data>
      <data key="d9">eth2</data>
    </edge>
  </graph>
</graphml>
```

Figure 26.   GraphML Network Topology

In this implementation the mapping module's capability is modeled by importing a properly generated GraphML file that represents a network topology. The network administrator accomplishes this by using the mapping drop-down menu to import the file that the awareness module uses to generate and display the topology. In the full-scale implementation the network administrator would still be able import a topology, but he or she could also select active and passive methods to scan the real-world network for the awareness and virtualization modules.

3. Awareness

The awareness module acts as the interface for the network administrator in addition to displaying the topology and other information. Our implementation allows the user to interact with a GUI driven menu system to initiate actions with the rest of the modules. The unique idea behind the awareness interface is that it presents the actual topology for which the network administrator is responsible and it also serves as the platform to conduct training. This means that the administrator gets realistic training on the actual tools used in tactical environments.

The first step in our implementation is loading the environment. The environment loads into a blank screen waiting for input from the network administrator. This interface uses a drop down menu on the side to represent each of the modules and a large canvas that is used to display the network topology. Nothing is automatically loaded. Figure 27 shows the starting interface.

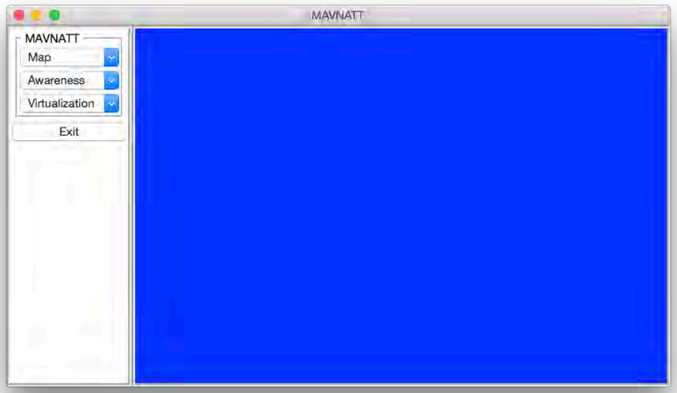


Figure 27. MAVNATT Implementation - Starting Interface

The prototype implementation allows the network administrator to import the network topology as discussed in the previous section. The network topology is parsed and then displayed on in the GUI as icons. Figure 28 shows the results after importing the topology.

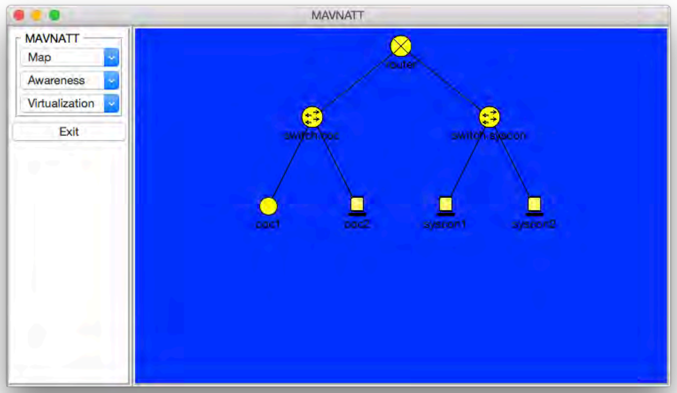


Figure 28. MAVNATT Implementation - Import Network Topology

The implementation graphically displays nodes as they are described in the imported GraphML network topology file shown in Figure 26. Icons are available to represent router, switch, computer, or unknown device types. The network administrator can then interact with the icons on the screen. This implementation only allows for moving the icons around the screen, but other actions can be implemented in the full awareness module, such as retrieving or setting the device type, device OS, interfaces, and IP addresses.

The awareness module should display network status or node health to the network administrator. The prototype of this implementation uses ping to demonstrate that each node is reachable. Figure 29 shows the results after the network administrator selects the ping status from the awareness drop-down menu.

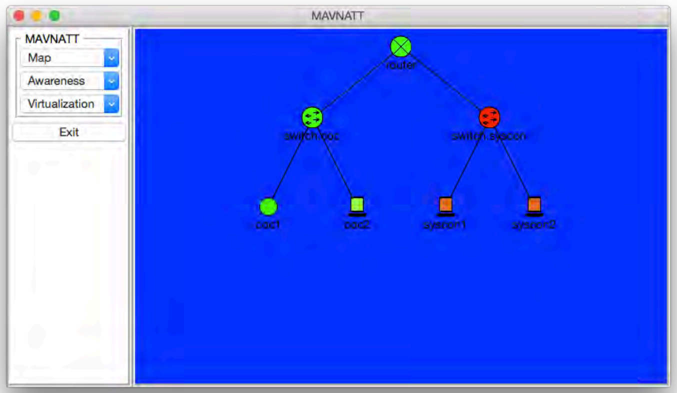


Figure 29. MAVNATT Implementation - Ping Status

The network administrator sees the result of the ping status test with the icons changing to green if the node was reachable, or red if the node was unreachable. Reachability tests are not path-aware in this implementation. In a fully capable awareness module we would expect to be able to get a greater level of device status and path-awareness with protocols like SNMP or WMI.

This is the point where MAVNATT steers away from a simplistic network-monitoring tool. In the final version of MAVNATT, the awareness module must achieve the ability to overlay a virtual topology on top of the physical topology. This will allow the network administrator to execute training scenarios concurrent with monitoring the network at a watch station. The network administrator currently only sees one topology, but the system should visually delineate real-world network faults from training induced virtual network faults, such as using different color schemes, to allow the network administrator to respond appropriately. Therefore, network faults could be induced at the virtual layer, while the physical layer remains healthy. This schema would also allow a network administrator to stop a training scenario and respond to a real-world fault, if necessary. Figure 30 graphically depicts this overlay architecture, which is separated into a live and virtual layer for viewing convenience.

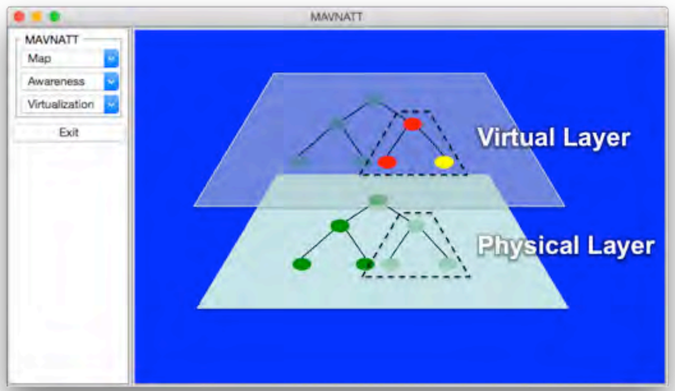
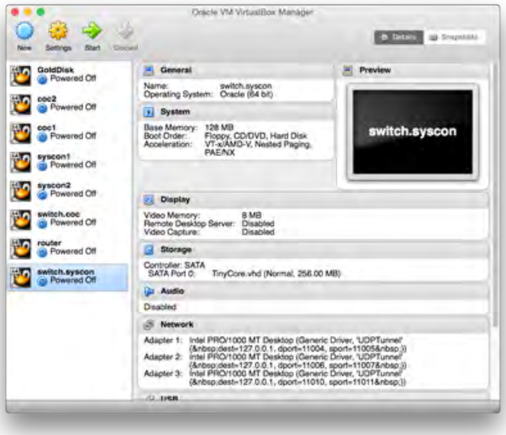


Figure 30. MAVNATT Implementation - Network Overlay

4. Virtualization

The virtualization module supports the virtualization layer discussed in the previous section. This module receives a file or graph object from the awareness module that represents the network topology. The virtualization module then creates a virtual topology that can be used to support training and evaluation of network administers on a virtual network that is a close representation of the physical network they are administering.

The prototype implementation demonstrates this capability by taking the network topology and creating a clone of a baseline configuration for each node. It also interconnects the nodes, based on the links reflective of the real world network, with a user datagram protocol (UDP) tunnel for each virtualized link. The virtualized network uses the VirtualBox hypervisor to pass network traffic between two virtual machines across the UDP tunnels without being broadcast to a live network. We manually set the IP address on each virtual machine for this implementation and the results validate that the nodes on each side of a link could ping each other. Figure 31 shows the VirtualBox user interface after the nodes are created.



```
GoldDisk        Powered Off
coc2            Powered Off
coc1            Powered Off
syscon1         Powered Off
syscon2         Powered Off
switch.coc      Powered Off
router          Powered Off
switch.syscon   Powered Off

Name:               switch.syscon
Operating System:   Oracle (64 bit)

Base Memory:   128 MB
Boot Order:    Floppy, CD/DVD, Hard Disk
Acceleration:  VT-x/AMD-V, Nested Paging,
               PAE/NX

Video Memory:              8 MB
Remote Desktop Server:     Disabled
Video Capture:             Disabled

Controller: SATA
 SATA Port 0:     TinyCore.vhd (Normal, 256.00 MB)

Disabled

Adapter 1:  Intel PRO/1000 MT Desktop (Generic Driver, 'UDPTunnel'
            (&nbsp;dest=127.0.0.1, dport=11004, sport=11005&nbsp;))
Adapter 2:  Intel PRO/1000 MT Desktop (Generic Driver, 'UDPTunnel'
            (&nbsp;dest=127.0.0.1, dport=11006, sport=11007&nbsp;))
Adapter 3:  Intel PRO/1000 MT Desktop (Generic Driver, 'UDPTunnel'
            (&nbsp;dest=127.0.0.1, dport=11010, sport=11011&nbsp;))
```

Figure 31. MAVNATT Implementation—VirtualBox and Virtual Machines

The "GoldDisk" entry in Figure 31 depicts the baseline machine that was cloned to generate the other nodes. As the virtualization module develops, more prefabricated baselines will become available for all devices, such as routers, switches, and servers. Attributes can be added to refine these devices and present a more accurate picture to the network administrator.

The main capability that the prototype implementation is missing is the synchronization between the newly created virtual nodes and the representation in the awareness module. When the virtualized topology and overlays are established, the network awareness module needs to communicate with the virtualization module for any training scenarios. The network administrator must be able to interact with the virtual nodes through the awareness module, and vice versa. This linkage is critical to the success of MAVNATT.

D. SUMMARY

This chapter presented a limited implementation of MAVNATT as a first step toward realization of the full capability envisioned for MAVNATT. The chapter first established the integrated prototype that used the framework to tie together the architecture modules. It then went through a step-wise flow of how the network administrator would use MAVANTT. The prototype successfully met all straw-man objectives. The implementation demonstrated starting the system, using the menu system to import a network topology, pinging nodes on the network to gain awareness, and virtualizing the topology. Implementation of fully functional components of MAVNATT remains for further research and development.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE RESEARCH

A. SUMMARY

This thesis was motivated by the necessity to improve the tools used to train network administrators in tactical environments while concurrently supporting network operations. We identified a training gap that exists for tactical network administrators and surveyed industry solutions that are used along the MAVNATT functional areas. These industry solutions fall into two major types of applications that apply to the MAVNATT functional areas: network monitoring suites and network simulators. Network monitoring suites and network simulators do not address the MAVNATT requirements as an integrated and lightweight system. Then the research creates the desired architecture and framework for MAVNATT, including the modules and components that support the design of MAVNATT. We then provide component-level and integrated prototypes to demonstrate desired capabilities and discuss the results for accuracy and completeness. The component prototypes demonstrate that MAVNATT is attainable at a functional level. We then discuss the implementation of MAVNATT by demonstrating concepts with an integrated prototype.

B. CONCLUSIONS

The conclusions for this thesis derive from the defined research objective and exploratory questions.

1. Research Objective

The primary research objective was to demonstrate that MAVNATT is an attainable objective. The component prototypes and the straw-man implementation via integrated prototype demonstrate that the functional capability exists to generate the desired MAVNATT system. With continued research and focused development on the various modules, a fully capable system can be achieved.

2. Exploratory Research Questions

This thesis provided six exploratory questions to support the overall research objective. Answering these questions provides additional insight into the research and allows more definitive conclusions and recommendations

- What are suitable formats to represent a network topology for importing, exporting, and sharing?

The research concluded that both GraphML and JSON were sufficiently capable of representing a network topology. The component prototype utilized the Python NetworkX package that allowed the files to be translated into a graph object that could be manipulated by MAVNATT.

- What are suitable libraries and application program interfaces (APIs) that can be utilized to create a GUI-based system that is capable of visually representing physical and virtual networks?

A wide variety of APIs support the development of such a system. Our implementation used the Python Tkinter GUI library, the Python NetworkX graphing library to import and export a GraphML document as a graph object, and the Python VirtualBox API. This combination successfully established an environment capable of visually representing physical and virtual networks.

- What are the methods of dynamically modeling a physical network to form a virtual network?

The integrated prototype allowed for the addition and removal of edge and node elements, demonstrating that dynamic modeling could be functionally achieved. In the context of the MAVNATT system, dynamic modeling, or the constant validation of the real-world and virtual topologies, would fall under the mapping and virtual modules.

- What are the methods of modeling network devices in a virtual environment?

Our research only modeled host devices as a network capability. It demonstrated that UDP tunneling could successfully connect two devices with minimal resources. As the modeling of network devices expands, this technique should be applied.

- Can this system be used to demonstrate scenario-based training?

The breadth of our research quickly exceeded original estimates and this question could not be addressed adequately under this study. Instead, the research was scoped to establish the architecture and framework for MAVNATT that we believe supports scenario-based training and provides the foundation for realization of the MAVNATT vision.

- Can this system work with current DOD tools?

Our implementation demonstrated that MAVNATT successfully operated on an IP infrastructure and that it could interoperate with current DOD IP-based tools. Further implementations of MAVNATT may require administrative permissions or other considerations to ensure the network security infrastructure allows it to operate on the network without being impeded by firewalls, host-based intrusion systems, and virus software. It is expected that a concise interface between the MAVNATT mapping function must be defined such that the tool can access the real network through a standard gateway, such as a firewall or network address translation server.

Additionally, our implementation utilized VirtualBox as the type-II hypervisor. The VirtualBox implementation supported our research by providing an open-source solution and access to its developer community. The DOD uses VMware virtualization software. We feel a VMware-based type-II hypervisor solution would be very similar in design and capability, but a type-I hypervisor would require some reengineering.

Provided these considerations are met, we do not perceive any interoperability concerns with DOD tools.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

The research, supported by the component-level and integrated prototypes developed as part of the study, indicate that a system like MAVNATT would increase network administrator capability by providing a system that supports concurrent operations and training. This thesis is intended to be a first stepping-stone for follow-on research and development, providing a foundation and conceptual design. Future researchers should improve upon the identified models, architecture, and framework. The following recommendations are provided to advance the MAVNATT initiative.

1. MAVNATT Architecture Future Research

a. Mapping Module

Further investigation is required with respect to identifying a network topology through active and passive network scanning methods and describing that network topology in a standard format. In addition, the following topics pertinent to network mapping are presented for future research:

- Credential-based methods, including with SNMP and WMI, to securely map the network topology while maintaining equivalent capability to industry standard network monitoring and mapping tools.
- Covert methods to map the network topology.
- Fingerprinting techniques for standard network devices.
- Network topology confidence levels to identify the accuracy of the discovered network topology to the actual topology.
- Methods to infer access control lists and network policies that affect the network topology and performance.

b. Awareness Module

Future research should continue to investigate how to visualize a network topology as discovered by the mapping module. Further study should also identify an overlay methodology to visualize the virtualized components on top of the live components for training and evaluation purposes. In addition, the following topics are presented for future research:

- Integrate a design capability into the awareness module that allows a network administrator to build and virtualize a network while not connected to a physical topology.
- Integrate a more capable image set that uses Scalable Vector Graphics (SVG) that could be integrated with the GraphML format.
- Integrate common network monitoring capabilities. Include a minimum set of network faults to be identified by the awareness module.
- Identify methods for implementing advanced protocols for path-awareness, route discovery, and forwarding table interrogation.

- Identify how to merge multiple network topologies from file formats into a single network topology. (Graph union operations)
- Identify how to compare multiple network topologies from file formats to identify differences. (Graph intersection and difference operations)
- Integrate common network management capabilities, such as SSH and TFTP, to assist in the management of both live and virtual devices.
- Integrate a web service for remote viewing of the MAVNATT interface.
- Provide the capability to integrate multiple MAVNATT instances with a shared view of the composite network.
- Integrate a logging capability to both the network operations and training aspects of the system.
- Represent dependencies, such as power and transmission devices, as layers in the awareness module.
- Identify how to deploy an environment built in the virtual environment, and evaluated within that environment, to the live network.
- Identify how to incorporate traffic capture, traffic generation, and traffic playback for network training and evaluation in MAVNATT.

c. Virtualization Module

This functional area should expand the investigation as to how to virtualize a network topology from a graph format and provide the capability to interact with virtualized entities for the purposes of training and evaluation. In addition, the following topics are presented for future research:

- Designate virtualization modes for training or evaluation. The training mode should support network administrator training by overlaying virtual devices with faults onto the physical device within the context of the awareness module. The evaluation mode should enable an interactive capability to evaluate network operator proficiency.
- Implement a capability within MAVNATT to support network planning and design.
- Determine the scalability of type-I, type-II, and cloud-based virtual solutions.
- Integrate channel emulation on virtual links to represent tactical transmissions and mobile adhoc networks.

- Integrate automatic provisioning software, like Puppet or Chef.
- Research simulating heavyweight network devices, like a Windows-based server, on lightweight virtual devices such as TinyCore Linux. This would reduce resource overhead and increase scalability during training and evaluation scenarios.

2. MAVNATT Framework Future Research

a. Network Topology Format Component

GraphML and JSON were successful in providing baseline data for the network topology. The prototypes were successful at utilizing these formats, but the virtualization component prototype identified a greater complexity of information that may need to be stored in the network topology format, such as router configurations, nested nodes and edges, and both live and virtual data for a single node or edge. The following areas need investigation:

- A MAVNATT-specific network topology format in XML.
- Libraries to abstract file input and output to ease programming of import and export of files from and to graph objects.

b. GUI Component

The Python component prototype successfully met all conceptual requirements. However, issues with the integrated prototype demonstrated the need to conduct further research in this area.

- Develop a Java Swing GUI prototype.
- Conduct usability and scalability testing with both the Python Tkinter and Java Swing integrated prototypes.

3. MAVNATT System Future Research

The following research topics refer to the MAVNATT system as a whole:

- There were many complications delivering a robust prototype for the MAVNATT system. In our research, we discussed GNS3, which is an open source network simulator. A future research topic may be to utilize GNS3 as the base of MAVNATT since it provides a pre-existing GUI interface, development module, and VirtualBox integration.

- Research integrating the MAVNATT final project into the Marine Corps systems planning, engineering and evaluation device (SPEED) and joint planning systems to support network planning and performance prediction.
- The final MAVNATT system should be formally integrated into the operational and training requirements outlined in the T&R manuals.

4. MAVNATT Employment Considerations

Throughout the research of this thesis, many different employment scenarios were discussed in which a MAVNATT-based system would provide advantages to improve current tactics, techniques, and procedures. Implementation of these employment scenarios requires a robust technology readiness level for MAVNATT. The following considerations could help to provide purpose for future development to achieve the necessary readiness level.

a. Network Planning and Validation Tool

MAVNATT could provide the capability to support planning and validation of network architectures in tactical environments. Network administrators could build a network topology in MAVNATT, virtualize it, evaluate it, and then deploy that network to the physical devices.

b. Integrated Network Training

MAVNATT could connect to other MAVNATTs representing a separate local area network or scope. Once connected, the root MAVNATT could deploy training scenarios that cross multiple networks and allow network administrators to execute cross boundary tactics, techniques, and procedures to communicate with external agencies. This integrated network training could be logged and evaluated to assess network administrator proficiency on a larger scale.

c. Cyber Operations

As cyber operations move to the tactical level of deployed operational units, offensive and defensive cyber operations become critical. MAVNATT provides a lightweight platform that is capable of mapping, visualization, and virtualization to support evaluation of both domains of cyber operations by projecting the impact of adversarial actions on friendly networks as well as providing a platform for predicting the efficacy of friendly actions on adversarial networks. Cyber operators could use MAVNATT to identify a cyber area of operations through passive or active mapping techniques from multiple entry points on a network. Once a network topology is identified, the topology could be virtualized and evaluated for vectors to a target. Scenarios could be run in the virtualized environment to model the attack vectors and determine success ratios. Furthermore, multiple views of the network topology could be compared and merged via back-channel communications to higher echelons for further evaluation and determination of a combined vector to increase likelihood of mission success.

LIST OF REFERENCES

[1] *Expeditionary Force 21*, U.S. Marine Corps, Washington, DC, 2014.

[2] *Joint Communication System*, Joint Publication 6–0, Joint Staff, Washington, DC, 2010.

[3] B. Williams. "The Joint Force Commander's guide to Cyberspace Operations," in *Joint Force Quarterly*, 2014, vol. 73, pp. 12–19.

[4] L. Pridmore, P. Lardieri, and R. Hollister. "National Cyber Range (NCR) automated test tools: Implications and application to network-centric support tools," in *Proc. AUTOTESTCON, 2010 IEEE*, Orlando, FL, 2010, pp. 1–4.

[5] R. Powell, T. Holmes, and C. Pie. "The information assurance range," in *TEA Journal*, 2010, vol. 31, pp. 473–477.

[6] *Military Occupational Specialties Manual*, NAVMC 1200.1, U.S. Marine Corps, Washington, DC, 2014.

[7] *Communications Training and Readiness Manual*, NAVMC 3500.56A, U.S. Marine Corps, Washington, DC, 2011.

[8] Internet Protocol Suite. (n.d.). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Internet_protocol_suite Accessed: May 12, 2015.

[9] How TCP/IP works. (2003, Mar. 28). Microsoft. [Online]. Available: https://technet.microsoft.com/en-us/library/cc786128(v=ws.10).aspx

[10] R. Siamwalla, R. Sharma, and S. Keshav. "Discovering Internet topology." *Unpublished manuscript*, 1998.

[11] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. "Avoiding traceroute anomalies with Paris traceroute," in *Proc. 6th ACM SIGCOMM Internet Measurement Conference*, Rio de Janeiro, Brazil, 2006, pp. 153–158.

[12] D. Harrington, R. Presuhn, and B. Wijnen. "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," STD 62, RFC 3411, Dec. 2002.

[13] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd ed. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[14] What is SNMP? (2003, Mar. 28). Microsoft. [Online]. Available: https://technet.microsoft.com/en-us/library/cc776379(v=ws.10).aspx

[15] Angry IP Scanner. (n.d.). Angry IP Scanner. [Online]. Available: http://angryip.org Accessed: May 12, 2015.

[16] Ipscan. (n.d.). Github. [Online]. Available: https://github.com/angryziber/ipscan Accessed: May 12, 2015.

[17] Nmap. (n.d.). Nmap. [Online]. Available: https://www.nmap.org Accessed: May 12, 2015.

[18] Nmap. (n.d.). Github. [Online]. Available: https://github.com/nmap/nmap Accessed: May 12, 2015.

[19] D. Mauro and K. Schmidt. *Essential SNMP,* 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., 2005.

[20] P. Moceri. "SNMP and Beyond: A Survey of Network Performance Monitoring Tools," 2004.

[21] SolarWinds screenshot. (n.d.). SolarWinds. [Online]. Available: http://cdn.swcdn.net/creative/v13.1/images/screenshots/products/NPM/Lg/En/NPM_11.0_Main-Screen-Shot_Base_Lg_EN.jpg Accessed: May 12, 2015.

[22] SolarWinds. (2015). *SolarWinds Network Performance Monitor Administrator Guide,* v11.5. SolarWinds Worldwide, LLC. [Online]. Available: http://www.solarwinds.com/documentation/orion/docs/orionnpmadministratorguide.pdf

[23] M. Qadir. "Comparative Analysis of two Open Source Network Monitoring Systems: Nagios & OpenNMS," Ph.D. dissertation, Blekinge Institute of Technology, 2010.

[24] Nagios. (n.d.). Nagios. [Online]. Available: https://www.nagios.org Accessed: May 12, 2015.

[25] Nagios. (n.d.). Sourceforge. [Online]. Available: https://sourceforge.net/projects/nagios/ Accessed: May 12, 2015.

[26] Nagios screenshot. (n.d.). Sourceforge. [Online]. Available: http://nagios.sourceforge.net/images/screens/new/home.png Accessed: May 12, 2015.

[27] OpenNMS. (n.d.). OpenNMS. [Online]. Available: http://www.opennms.org Accessed: May 12, 2015.

[28] OpenNMS. (n.d.). Github. [Online]. Available: https://github.com/OpenNMS/opennms Accessed: May 12, 2015.

[29] Hypervisor. (n.d.*). Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Hypervisor Accessed: May 12, 2015.

[30] Developers, VirtualBox. (2015, May 12*). VirtualBox Screenshot*. [Online]. Available: https://www.virtualbox.org/raw-attachment/wiki/Screenshots/win7.png

[31] Oracle VM VirtualBox user manual. (n.d.). Oracle. [Online]. Available: https://www.virtualbox.org/manual/UserManual.html Accessed: May 12, 2015.

[32] VirtualBox. (n.d.). Oracle. [Online]. Available: https://www.virtualbox.org/browser/vbox/trunk Accessed: May 12, 2015.

[33] VMware Fusion. (n.d.). VMware. [Online]. Available: https://www.VMware.com/products/fusion Accessed: May 12, 2015.

[34] VIX API Documentation. (n.d.). VMware. [Online]. Available: https://www.VMware.com/support/developer/vix-api/ Accessed: May 12, 2015.

[35] VMware Fusion 7 Screenshot. (n.d.). VMware. [Online]. Available: https://www.VMware.com/files/images/screens_fusion/f7/vmw-scrnsht-fusionpro-cloning.jpg Accessed: May 12, 2015.

[36] Kernel Virtual Machine. (n.d.). Linux KVM. [Online]. Available: https://www.linux-kvm.org/page/Main_Page Accessed: May 12, 2015.

[37] GNS3. (n.d.). GNS3 [Online]. Available: https://www.gns3.com Accessed: May 12, 2015.

[38] Common Open Research Emulator (CORE). (n.d.). U.S. Navy Research Lab. [Online]. Available: http://www.nrl.navy.mil/itd/ncs/products/core Accessed: May 12, 2015.

[39] J. Ahrenholz, "Comparison of CORE network emulation platforms," in *Proc. IEEE MILCOM Conference*, San Jose, CA, 2010, pp. 864–869.

[40] Netkit. (n.d.). Netkit [Online]. Available: http://wiki.Netkit.org/index.php/Main_Page Accessed: May 12, 2015.

[41] Neo4J. (n.d.). Neo4j [Online]. Available: http://www.neo4j.com Accessed: May 12, 2015.

[42] J. Webber. "A programmatic introduction to neo4j," in *Proc. 3rd annual conference on Systems, Programming, and Applications: Software for Humanity*. ACM, 2012, pp. 217–218.

[43] JSON. (n.d.). JSON. [Online]. Available: http://www.json.org Accessed: May 12, 2015.

[44] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta. “Comparison of JSON and XML Data Interchange Formats: A Case Study,” *Caine,* 2009, pp. 157–162.

[45] The GraphML File Format. (n.d.). GraphML [Online]. Available: http://graphml.graphdrawing.org Accessed: May 12, 2015.

[46] L. Prechelt. “An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl,” *IEEE Computer* 33, vol. 10, 2000, pp. 23–29.

[47] D. Wu, L. Chen, Y. Zhou, and B. Xu. “A metrics-based comparative study on object-oriented programming languages.”

[48] S. Nanz and C. Furia. “A comparative study of programming languages in Rosetta Code.” *arXiv preprint arXiv:1409.0252,* 2014.

[49] About the JFC and Swing. (n.d.). Oracle. [Online]. Available: http://docs.oracle.com/javase/tutorial/uiswing/start/about.html Accessed: May 12, 2015.

[50] Tkinter. (n.d.). Tkinter [Online]. Available: https://wiki.python.org/moin/TkInter Accessed: May 12, 2015.

[51] Qt. (n.d.). *Qt* [Online]. Available: http://www.qt.io Accessed: May 12, 2015.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California